1979

A tactical system emulator for a distributed micro-computer architecture.

Guillen, Luis A.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

A TACTICAL SYSTEM EMULATOR
FOR
A DISTRIBUTED MICRO-COMPUTER ARCHITECTURE

by

Luis A. Guillen

June 1979

Thesis Advisor: Uno Kodres

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)

A Tactical System Emulator for a Distributed Micro-Computer Architecture

5. TYPE OF REPORT & PERIOD COVERED
Master's Thesis
June 1979

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)

Luis A. Guillen

8. CONTRACT OR GRANT NUMBER(s)

9. PERFORMING ORGANIZATION NAME AND ADDRESS

Naval Postgraduate School
Monterey, California 93940

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS

Naval Postgraduate School
Monterey, California 93940

12. REPORT DATE
June 1979

13. NUMBER OF PAGES
324

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

Naval Postgraduate School
Monterey, California 93940

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

EMULATOR, MICRO-COMPUTER, OPERATING SYSTEM,
REAL-TIME SYSTEM, MULTIPROCESSING, TACTICAL SYSTEM

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An emulator is contructed in order to study the behaviour of a Tactical System in a distributed micro-computer architec_ ture environment. This emulator represents Tactical Systems as a set of periodic and demand scheduled functional module processes that colaborate with each other.

A special purpose operating system was implemented for the distributed micro-computer architecture that supports the emu-

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |
(Page 1)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

lator. It includes processor managment and system wide input/output capabilities.

Approved for public release; distribution unlimited

A TACTICAL SYSTEM EMULATOR FOR A DISTRIBUTED MICRO-COMPUTER ARCHITECTURE

by

Luis A. Guillen
Lieutenant (JG), Peruvian Navy
B.S., Peruvian Naval Academy, 1974

Submitted in partial fulfillment of the
requirement for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
June 1979

ABSTRACT

An emulator is constructed in order to study the behaviour of a Tactical System in a distributed micro-computer architecture environment. This emulator represents Tactical Systems as a set of periodic and demand scheduled functional module processes that colaborate with each other.

A special purpose operating system was implemented for the distributed micro-computer architecture that supports the emulator. It includes processor management and system wide input/output capabilities.

TABLE OF CONTENT

ACKNOWLEDGEMENT

I must thank professor Uno Kodres, my advisor, for his support and constant dedication and professor Roger Schell for his comments and guidance. To them I owe most of the ideas of the thesis.

I also thank Mr. Bob McDonnell and Mr. Michael Williams, from whom I have learned so much during the implementation of the system. Finally I thank my wife, Marcela, for her patience.

I. INTRODUCTION.

A. MOTIVATION.

LSI technology has brought to the market new micro-computers which meet most of the requirements needed for tactical system development. They are highly reliable (hardware), small, easy to mantain, don't need much power, have a powerful instruction set and also, because the great acceptance within the general market, they are relatively cheap and have considerable software support. The question is whether new architectures built from these computers will perform in a satisfactory manner within the real-time constraints imposed by tactical systems.

Tactical Systems are mostly dedicated systems, that is, they are designed to accomplish repetitive computations over the same algorithm. They are, also, functionally oriented, and most of the time the designer has a very good idea about the amount of execution time and memory space needed by each function before implementation takes place.

Tactical Systems can be viewed as a set of functional modules related by communication links. With this in mind, a tactical system could be mapped into a di-graph, where a node would represent a functional module and an arc the information crossing from one module to another.

As an example, take the di-graph in Figure 1. Nodes 1,2,3,4 and 5 represent functional modules which perform certain predetermined functions needed by the tactical

system. Arcs (a),(b),(c) and (d) represent information crossing from one module to another.

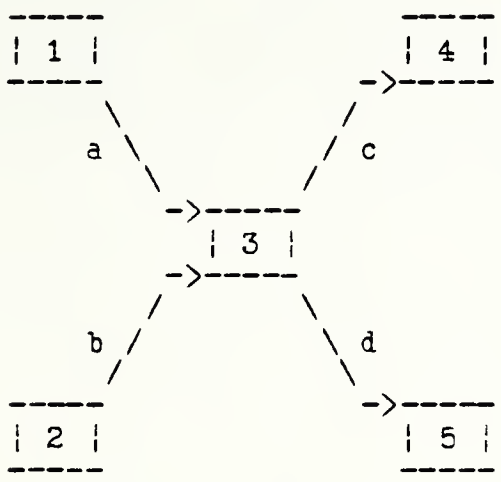

FIGURE 1. Example of a Di-graph

We must now observe that each node or set of nodes can be mapped into a process, where a process is characterized by an execution point and an address space, and each arc can be mapped into an interprocess communication message. With this mapping sequence we transform the original tactical system into a computer system design.

The task of partitioning the diagraph so as to define the process set is not addressed in this thesis. Reference [4] reviews the published literature on graph partitioning.

Early tactical system implementations concentrate the computation effort in one relatively large, fast computer. In doing so, these implementations arrange the processing sequence into a totally ordered set. i.e. in our previous example, the processor would be used successively by

processes 1,2,3,4,5 and back to 1. The processor needs to be fast enough so it can be used by all the processes within the real-time constraints without diminishing the performance of the system. Sometimes these systems work based on a pooled algorithm in which the processes which do not need the immediate use of the processor are skipped until the next round. Even then, the processor must be such that it can handle the maximum load.

A different implementation approach can be taken. We could concentrate our effort in distributing the processes among several slower computers which collaborate with each other in pursuing the same system goals. Our main concern in distributing the processes among several computers is the resulting inefficiencies introduced by the distribution:

1) Greater communication problems between computers.

2) Duplication of data and programs.

3) Imbalance in execution time and program size among the processors.

Reference [3] explains a data flowgraph technique that allows us to explicity determine the number of data elements which must be communicated between processes. The flowgraph analysis allows us to conclude if our processes could be carried out in different computers completely independent without creating communication problems.

Given that we have followed the last approach, we are interested in finding out whether we can meet the real-time requirements imposed by the tactical systems. Following our

previous example, we can examine the idea using Figure 2.

CPU A is the fast processor used in the centralized implementation. Assume we have two slower processors (CPU 1 and CPU 2) for our distributed approach. Given that we have determined that processes 1 and 2 and then 4 and 5 can be executed , we want to find if the time lapse in which the five processes are executed by CPU 1 and CPU 2 meet the real-time requirements.

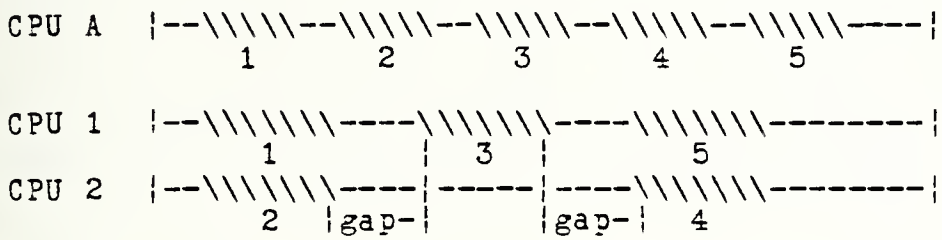

FIGURE 2. Time lines for centralized and distributed approaches.

In most typical tactical systems, we can estimate the execution time required by each process, as was mentined before. The critical point in finding the time frame in which the system is going to work is the interprocess execution gaps and how much they diminish the overall performance.

A hardware architecture has been proposed in Reference [3]. In this architecture the system is built from identical single board processors such as the INTEL SBC80-20 or the Texas Instruments TM 9900. The boards are connected by the INTEL MULTIBUS, TI TILINE or the Digital Equipment's UNIBUS. A group of single board computers connected on a parallel bus is called an 'affinity group'.

A Real-Time Operating System for Single Board Computer Based Distributed Naval Tactical Data System was developed at the Naval Postgraduate School (see Reference [2]) in order to support the implementation of tactical systems, on an architecture of independently operating single board processors.

In order to determine if such architectures can effectively support tactical processing, the results of analysis should be verified by a more realistic emulation of the system. It is the purpose of this thesis to implement an emulation system based on a multiple single board computer architecture and the real-time distributed operating system [Ref. 2].

B. DESIGN OBJECTIVES.

As mentioned before, a Tactical System could be mapped into a system of independent processes related by interprocess communication messages. The goal of this thesis is to create a Tactical System Emulator in order to study the behaviour of such a system in a distributed microcomputer architecture.

The Tactical System Emulator intends to be a tool for helping the tactical system designer in allocating the different processes, into which his system has been partitioned, between the processors available in a proposed microcomputer network. By using the emulator, the system's designer can identify potential communication bottlenecks

before the system is fully implemented.

The Tactical System is viewed for this purpose as a set of functional modules that interact with each other through messages. There are two kinds of functional modules: periodic and demand. Periodic functional modules become active every predetermined interval of time. They are triggered by the system and are usually in charge of input/output functions. Demand functional modules become active upon receipt of a message. This message can be sent by periodic as well as by demand functional modules.

The Emulator replaces every periodic functional module with a dummy periodic module and every demand functional module with a dummy demand module. These dummy modules are not expected to perform the required function but to consume execution time in the same way as if they were doing it. They are also expected to transmit dummy messages of the same length and destination as the ones sent by the real functional modules they are replacing.

At execution time, the Emulator would save statistics about the behaviour of the system with emphasis on the interprocess communication delays.

The emulator would also provide a dynamic tactical system´s definition tool. This definition tool will allow the user to define the real Tactical System´s parameters and then will distribute this information among the Single board computers, so as to prepare the emulator data structure for the emulation. In other words, the user will interact with

the emulator system in preparing the emulation environment. The emulator will require system parameters such as the CPU time needed by each functional module, number of messages , destination and length of each message, number of processors needed and so on.

It would be also necessary that the emulator allow the user to redistribute the modules between processors and to modify the emulation parameters dynamically.

C. THESIS BODY.

The architecture of the distributed system of SBC80-20 micro-computers was chosen because the availability of the single board computers in the micro-computer laboratory. The availability of the system's programming language, PLM80, and the existance of a previous thesis which implements a real-time operating system for the same architecture (see Reference [2]) were equally important reasons for choosing this architecture.

Tailoring this operating system to meet the emulator needs and to run in the choosen architecture, took most of the thesis effort. Chapter II and III explain the characteristics of the system created to support the emulator and the implementation details.

Chapter IV describes the emulator as it was conceived and implemented. Appendix C explains how to use the emulator.

II. DISTRIBUTED SYSTEM.

A. GENERAL IDEA.

Real-time applications push computer and programming technology to its limits (and sometime beyond). A real-time system is expected to monitor simultaneous activities with critical timing constrains continuosly and reliably. The consequences of system failure can be serious.

Real-time systems must achieve the ultimate in simplicity, reliability, and efficiency. Otherwise one can neither understand them ,depend on them, nor expect them to keep pace with their environment.

To make a real-time system efficient will probably require the design of computer architectures tailored to particular applications. Real-time systems have these characteristics:

(1) A Real-time System interacts with an environment in which many things happen simultaneously at high speeds.

(2) A Real-time system must respond to a variety of asynchronous requests from its environment. The system can´t predict the order in which these request will be made but must respond to them within certain time limits. Otherwise, input data may be lost or output data may lose its significance.

(3) A Real-time System controls a computer with a fixed configuration of processors and peripherals and performs (in most cases) a fixed number of concurrent task in its

environment.

(4) A Real-time System never terminates but continues to serve its environment as long as the computer works. (The occasional need to stop a real-time system, at the end of an experiment can be handle by an ad hoc mechanism, such as turning the machine off or loading another system into it.)

What is needed then for real-time applications is the ability to specify a fixed number of concurrent tasks that can respond rapidly to asynchronous requests. The system proposed here is a real-time system with the following characteristics:

(1) A Real-time System consist of a fixed number of concurrent processes that are started simultaneously and exist forever. Each process can access its own variables as well as shared information.

(2) Processes can interchange information by sending messages to each other. This way a process can request services from another process. This is the only form of interprocess communication.

(3) Processes are synchronized by means of interprocess communication messages.

B. ARCHITECTURE.

The proposed architecture consist of a set of identical single board computers and read/write memory boards connected by a time multiplexed data bus.(see Figure 3.)

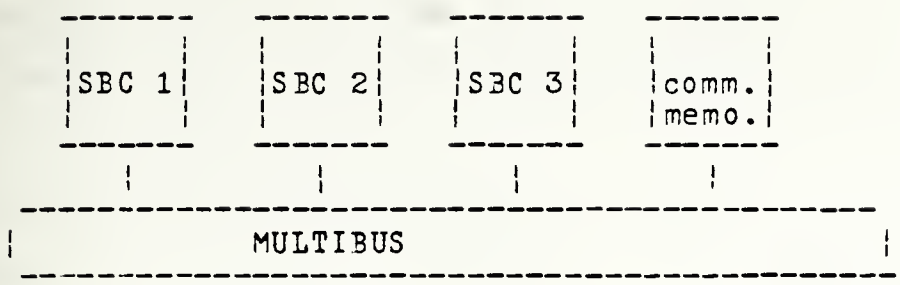

FIGURE 3. An "Affinity Group".

Each Single Board Computer (from now on named SBC) is able to communicate with the others by means of a message buffer located in common memory (the memory boards). When one SBC wants to transmit some information to another, it would store a message with the information in common memory and then the receiver will pick it up from there.

The system to be implemented could be defined as a special purpose operating system. It provides interprocess communication, in the form of message routing, and process scheduling. It also provides system wide input/output facilities. Memory and Information management are not considered necessary for this class of tactical operating system, so these functions must be provided by the user.

The Operating System recognizes three kinds of tasks: Priority task, Message task, and Periodic task. All user´s applications must be tailored to fit into these categories.

Priority task are scheduled for system events. These usually cover the input/output and real-time clock computational needs. A zero count in a system count-down clock, an interrupt or a system reset are examples of event that would produce the scheduling of one of these task.

Message tasks are scheduled upon receipt of a message for them. These are usually defined by the user and carry most of the computations.

Periodic tasks are scheduled at the end of a time-lapse defined by its period. A periodic task is defined with an entry point, and an interval of time (period). The system keeps track of the next activation-time and the state of the task.

C. THE EXECUTIVE.

Each SBC is driven by an executive which resolves priorities between system task based on the algorithm described by the flowchart in Figure 4.

Priority tasks have the highest priority. Message tasks have the next highest priority. Periodic tasks have the next highest priority and background tasks proceed at the lowest priority level.

The Executive must check for messages comming from two different sources. One is the SBC buffer used by the system as an immediate message buffer. The other is the System buffer used as a system wide message buffer. Messages directed to messages task allocated in the same computer as the source will only use the SBC buffer. If the directed message task is not allocated in the same computer the message will be put in the system buffer to be picked up by the respective executive.

Periodic task initiation is based on the content of the

real-time system clock. Each periodic task is associated with an activation time. When this time is lower or equal than the real-time clock the task will be scheduled. The real-time clock must be system wide so that the reference for all the executives will be the same.

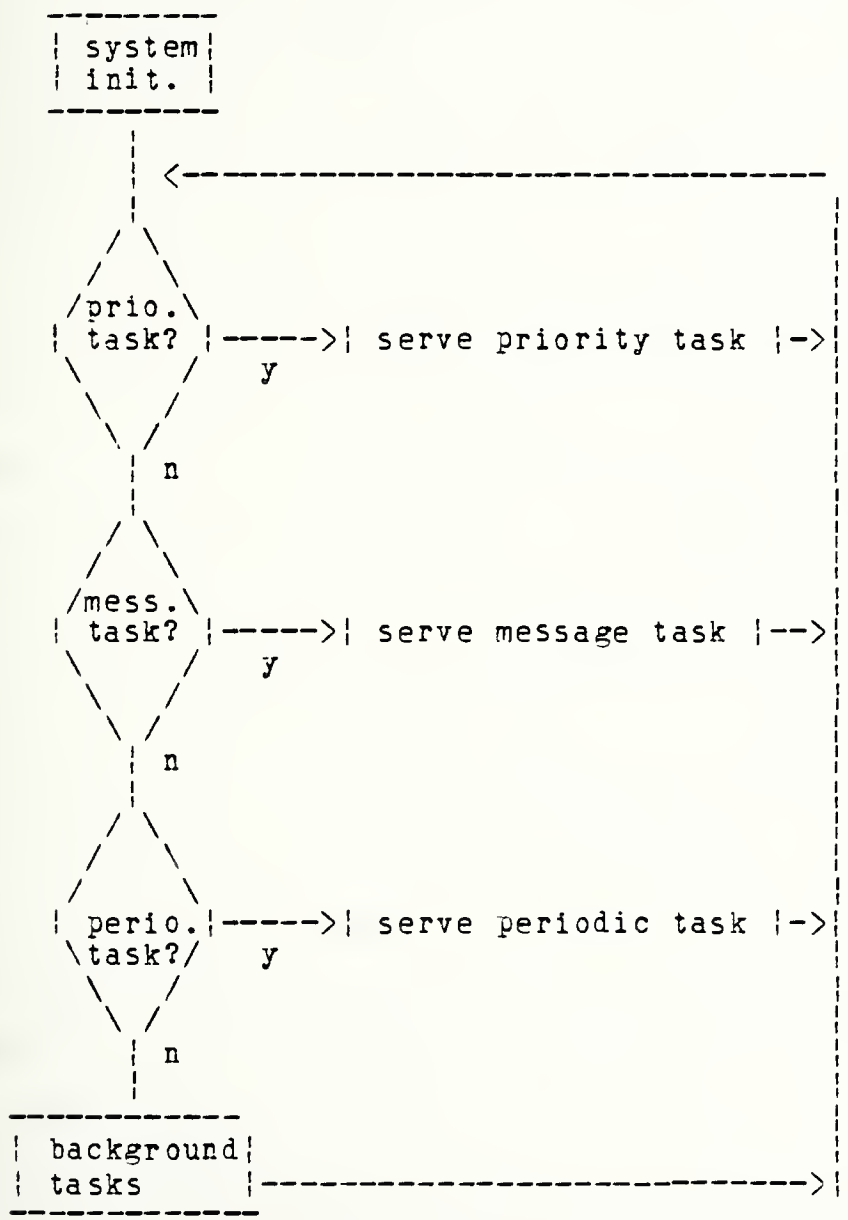

FIGURE 4. Executive Flowchart.

D. INPUT/OUPUT.

Each SBC must be able to handle its own input/output resources. The presence of data to be transmitted or received from an external device will be considered as a system event able to be identified locally by the SBC to which the device is connected.

This input/output event will produce the scheduling of a corresponding priority task which would be able to receive or transmit the data.

All input/output procedures then would be made at this level. The interchange of data will be between the external devices and locally allocated input/output buffers.

A combination of priority and message tasks should be used for serving the interrupt driven input/output functions. Interrupt driven I/O must be used rather than dedicated I/O in order to allow other tasks to be served while the communication interchange is in progress and contributed to meet the real-time constrains as needed.

For the serial I/O interface constructed, an input (priority) task will collect the input data in a buffer. When the input task receives an end of data notification, it will empty the buffer by sending a message to the task which uses the data.

A virtual system console, able to be attached to any one of the processors is created. It provides the user with a powerful tool for constructing user interface software. Ad hoc system routines like PRINT$SYNC and CON$INPUT$REQ, for

input/output to this console, are offered to the user in order to facilitate the control of the virtual device.

Because the characteristics of the system, an input operation is not a function but a request for an input message string. In the case of the system console, the message will come from the real device attached to it.

D. THE MONITOR.

Special software was created to allow the user to interact with the system in order to examine and modify its parameters.

As defined before, each SBC has its own local memory which the system uses to store data and certain parameters. In the development of applications it is very important to have access to the data not only for debugging purposes but to be able to recognize transient states that can help in the improvement of the application.

The system monitor contains commands for displaying, filling, and modifying memory, for the transmission of messages from the console and the allocation of this console to any one of the computers. It interacts with the input/output task for communicating with the user and is mainly composed of message tasks.

A special feature includes triggering the monitor when control gets to a defined execution point. This event, requested by the monitor with a command, will schedule a priority task which in turn will schedule the monitor of the

computer which got to this execution point.

E. USING THE SYSTEM.

This special purpose executive operating system has not been created with the idea of a stand alone system but as a tool that must be used in conjuction with the program development system that helps the user to set up the initial state.

Each SBC will have in its read only memory a copy of the executive, a serial input/output handler, a parallel output handler, an interrupt handler, a monitor and a set of system routines that will help the user to control the on-board system resources and the communication between processes.

The system data structures include an address vector for priority task entry points, an address vector for message task entry points and status, and an information vector for each periodic task containing status, entry point address, interval time address and next activation time. This data, in conjunction with a real-time clock, a message buffer and other system parameters, which will be described in Chapter III are allocated in common memory. The system provides system routines for controlling these data structures.

Message tasks are created by defining a message entry point and an initial status (active or inactive). This can be done before the distributed system initialization, provided the initialization message is accepted. Any periodic task can be created, suspended and its period

modified using system routines. In the same way priority task can be controled.

In addition of the purposes mentioned previously, a variety of system routines have been created for helping the tactical system designer. The user would only need to properly link his programs with the system's public reference in order to use them.

Special care must be taked in the linking and locating process at implementation time, because there is no mechanism for memory management. The user himself would be responsible for the proper location of his programs.

Each SBC can control (essentially by multiprogramming) up to eight priority tasks, eight message tasks, and eight periodic tasks.

III. DISTRIBUTED SYSTEM IMPLEMENTATION.

A. ENVIRONMENT.

The system has been built from three INTEL SBC80/20-4 Single Board computers and four 16K RAM random access memory boards. The description of the SBC and the MULTIBUS by which these boards communicate with each other can be found in Appendix A and B respectively. Detailed information about the computer is found in Reference [5].

In order to program the software for the system, the INTELLEC MDS Microcomputer Development System was used. A complete description of this system can be found in Reference [6].

Software used for the development of the distributed system, as well as for the emulator, include ISIS-II Operating System and PLM80 compiler. Information about these software products can be found in References [7,8].

The distributed system's memory organization can be seen with the help of Figure 5.

- ISIS-II operating system uses the lowest 12K bytes of common memory. Locations from 0000H to 2FFFH.

- MDS monitor uses the highest 2K bytes of common memory. Locations from F800H to FFFFH.

- The distributed system code uses the lowest 8K bytes of on board memory, locations from 0000H to 1FFFH. This memory segment contains the Executive, interrupt handler, messages handler and input/output handlers, the system

initialization software, the monitor and all the system routines. One copy for each single board computer.

- The Distributed System interrupt vectors and local variables use on board SBC memory from 3000H to 38FFH. On board memory from 3900H to 3FFFH is available to the user.

- The Distributed System general data structures are allocated in common memory from F400H to F7FFH.

- Locations from 4000H to F3FFH are free for use. These are 45K bytes of common memory available to the user.

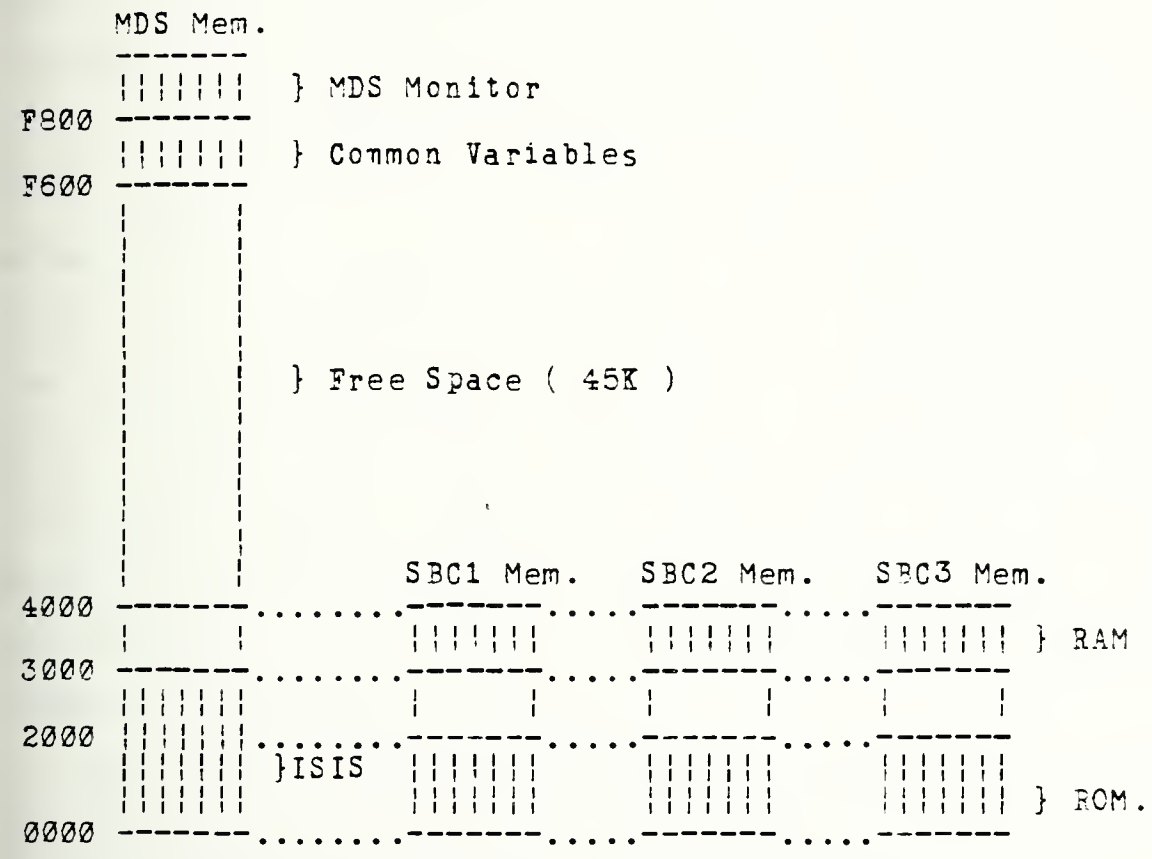

FIGURE 5. System´s Memory Organization

On board SBC memory shadows the corresponding locations on common memory, so that the SBC can´t access common memory from 0000H to 1FFFH or from 3000H to 3FFFH. In the same way,

a SBC's is protected against other processors trying to access its local memory.

The MDS System provides a front panel board which, besides working as an interface between the front panel switches and the MDS processor and other functions unimportant in this context, serves as a priority resolver for the Bus priority logic. It resolves bus contention for up to eight master modules. The logic monitors eight bus request (BREQ) lines, arbitrates all request in parallel, and controls eight bus priority enable (BPRN) lines. Only one BPRN line associated with the highest priority module which is requesting use of the bus is enabled, thus allowing that board to become bus master. The front panel board also generates the 9.8 MHz bus clock (BCLK/) signal which provides a timing reference for the bus control section of the various master modules. This board is the only other board used after the Distributed System has been initialized. Another feature of the front panel board, namely the interrupt switch logic for interrupt number six is used as a "warm boot" for reinitializing the system (without turning the power off).

The ISIS-II Operating System is used to develop the programs for the Distributed System applications, as was the case with the Emulator programs.

B. SYSTEM INITIALIZATION.

Several steps must be taken before the executive takes

control of the system. They can be separated into three steps.

The first one is executed under ISIS-II Operating System, working with the standard MDS processor and system console; its purpose is to link the user´s application programs within the Distributed System.

The second step will put the MDS processor into a HALT state and will activate the SBC´s which are needed as part of the system.

The third step will be taken by each SBC, provided it has been activated, and its purpose is to prepare the local data structures and resources for the new job.

1. First Step

The user can make use of several system routines, which will be described in detail later. In order to do so, he only needs to declare them external variables and then reference them as indicated. Remember that the code belonging to these routines is already present and has been allocated in on board SBC memory. Because of the characteristics of the hardware, the memory cycle in this memory is free from bus contention and hence the use of these routines is highly recommended over similar ones created by the user.

To get to the system routines, it would be sufficient for the user to link his previously compiled code with the public reference of the Distributed System code.

This would resolve any external reference and would direct the processor to the corresponding routine in local memory.

The user must be sure that the initial state of the message task vector address is as he wishes. The message task vector address is one of the system wide data structures located in common memory. Its name is MSGMODADDRESS and is located at MSGTBLADR (F660H).

The program MOD$PRO.SRC provides two procedures for initializing this structure: CLEAR$MOD, which erases the entire vector and ENTER$MOD(#,Address), which initializes a message task by defining the message number to be used and the entry point address of the task. Computer 1 serves message tasks numbered between 1 and 7; computer 2 , message tasks numbered between 9 and 15 and computer 3, message tasks numbered between 17 and 23. Numbers 0, 8 and 16 are reserved for the Distributed System itself.

2. Second Step

Upon system reset, all three SBC's begin to look at a special system wide variable called LOADSBC. When LOADSBC takes the value equal to the number with which the SBC identifies itself, it is the time for waking up. First thing to do for the SBC would be to check a second vector variable called START in the position corresponding to its identification number (1,2 or 3). If the computer can find the same number in this position, then it proceeds to initialize itself, otherwise it keeps checking START every

second, forever.

The program called RUN, which executes under ISIS-II, receives as input the number of the SBC´s which the user is willing to use. SBC´s are identified by the numbers 1, 2 and 3. Upon receipt of the input, it sets the corresponding positions of START and sets variable LOADSBC to 1. LOADSBC will be incremented by computer 1 and then by computer 2, in case it has been requested. The program then puts the MDS processor in HALT state.

The program also provides for a "soft boot" reset routine with interrupt number six. This interrupt would be used as a reinitilization interrupt by the Distributed System.

3. Third Step

Several steps must be taken by each SBC before giving control to the executive. They include the initialization of:

(a) system wide data structures.

(b) executive parameters.

(c) interrupt controller.

(d) counters.

(e) serial and parallel I/O interfaces.

(f) interrupt mask.

We should see these steps in more detail.

a. System data structures initialization.

Before the executive takes control, the

following must be accomplished with the data:

(a) System wide message buffer, EXTMSGBUFFER must be empty and the corresponding pointers and flags set to zero.

(b) The real-time clock RTC set to zero time, and its semaphore, CLOCK set to free.

(c) Immediate message buffer, MSGBUFFER must be empty and the corresponding pointers and flags set to zero.

(d) The address vector for priority task and periodic task (PRIORLIST and PERLIST) set to null.

(e) MSGENTØ must be entered as a message task in the corresponding number (Ø, 8 or 16). This task groups the entry points of the I/O interfaces and the monitor.

(f) All control parameters and variables set to zero.

All the step listed above are accomplished by procedure SETEXDATA in INTMSG.

b. Executive parameters initialization.

The system provides an initialization message, numbered Ø, and several system messages for controlling the system console and the printer. Initially, the system console and printer will be defined in SBC 1, so these messages will be directed to message task number Ø. These messages are numbered from 1Ø to 25 and are initialized by procedure SETEXMSGS in INTMSG.

The procedure SETEXMSGS will also allocate an initialization message for each user message task defined

for the SBC.

c. Interrupt controller initialization.

When the processor receives an interrupt signal it checks the location of the corresponding interrupt handler in an interrupt vector previously defined. These interrupt vectors must be initialized and their location and characteristics communicated to the interrupt controller hardware. This is done by procedure SETEXINTE in INTMSG.

d. Counters initialization.

The INTEL SBC80/20 hardware has three counters, numbered 0, 1 and 2. Counter 0 is used to update the real-time clock, RTC. Counter 1 is used for a special mechanism which will trigger a user defined procedure at the end of the required count. Counter 2 is used for baud rate generation for serial I/O communication interface.

All three counters must be programmed to accomplish the purpose they have been assigned to do before they are used.

e. Input/Output interface initialization.

Serial and Parallel I/O interfaces are initialized by procedures SEIO$START and PAIO$START respectively. In general they must set the hardware for the proper communication protocol, clear the I/O buffers and pointers, and introduce the corresponding I/O priority task into PRIORLIST.

f. Interrupt mask initialization.

The interrupt mask is saved in system variable

INTMASK. In case of computer 1, the mask would be set to allow counter Ø to produce an interrupt for RTC updating, and to enable interrupt six in order to reinitialize the initialization sequence whenever the user wants to push the front panel switch. Besides this, all SBC interrupts three, four and five are enabled for I/O purposes.

Counters, I/O interface, and interrupt mask initialization is accomplished by procedure EXSTART in INTMSG.

C. INTERRUPTS.

1. Hardware characteristics.

The INTEL SBC8Ø/2Ø-4 interrupt controller logic consist of Intel's 8259 Interrupt controller device and a jumper pad that allows the user to connect any of 27 possible interrupt request to the 8259's eight interrupt priority inputs. The 8259 resolves priority among all eight levels according to an algorithm which is program selected by the user. The normal interaction of the 8259 with the CPU is as follows:

(a) one or more of the interrupt request lines (IR7-Ø) is raised high signalling to the 8259 that the peripheral equipment is demanding service.

(b) The 8259 accepts these requests, resolves the priorities, and sends an INT to the 8Ø8Ø CPU.

(c) The 8Ø8Ø CPU acknowledges the INT and responds

with an INTA/ pulse.

(d) Upon receiving the INTA/ from the CPU group (8238), the 8259 will release a CALL instruction code (11001101) onto the 8-bit Data Bus thougth its D7-0 pins.

(e) This CALL instruction will initiate two more INTA/ pulses to be sent to the 8259 from the CPU group (8238).

(f) These two INTA/ pulses allows the 8259 to release its preprogrammed subroutine address onto the Data Bus. The lower 8-bit address is released at the first INTA/ pulse and the higher 8-bit address is released at the second INTA/ pulse.

(g) This completes the three byte CALL instruction released by the 8259. The in-service register (ISR) is not reset until the end of the subroutine when an EOI (End of interrupt) command is issued to the 8259.

The 8259 accepts two types of command words generated by the CPU for programming purpose:

a. Initialization Command Words (ICWs):

Before normal operation can begin, each 8259 in the system must be brought to a starting point--by a sequence of 2 or 3 byte commands timed by WR/ pulses.

b. Operation Command Words (OCWs):

These are the command words which command the 8259 to operate in various interrupt modes. These modes are: Fully nested mode, Rotating priority mode, Specific priority mode and polled mode.

The 8259 will operate in the fully nested mode after the execution of the initialization sequence without any OCW being written. In this mode, the interrupt requests are ordered in priorities from Ø to 7. When an interrupt is acknowledged, the highest priority request is determined and its address vector placed on the bus.

Whenever a command is issued with AØ = Ø and D4 = 1 this is interpreted as initialization command word 1 (ICW1), and this initiates the initialization sequence. During this sequence, the following occurs automatically:

(a) The edge sense circuit is reset, which means that following initialization, an interrupt request (IR) input must make a low to high transition to generate an interrupt.

(b) The interrupt mask register is cleared.

(c) IR 7 input is assigned priority 7.

(d) The special mask mode flip-flop and the status read flip-flop are reset.

Initialization command word 2 must be output right after ICW1. ICW1 provides two control bits and two or three of the interrupt CALL address bits. ICW2 provides 8 of the CALL address bits.

The 8 requesting devices have 8 addresses equally spaced in memory. The addresses can be programmed at intervals of 4 or 8 bytes: the 8 starting locations therefore occupy 32 or 64 bytes of memory respectively.

2. Interrupt Configuration.

There are two major consideration in configuring the interrupt structure on the SBC 80/20:

(a) The connection of external and on-board interrupt requests to the eight interrupt priority level inputs (IRØ-IR7) on the 8259.

(b) The selection of a priority resolution algorithm.

The priority resolution algorithm is selected by programming the 8259 as mentioned before. The assignment of interrupt requests to interrupt priority level inputs is made by connecting the appropiate jumper pins.

The 8259 was programmed in the fully nested mode for the Distributed System. Interrupt Ø has the highest priority and interrupt 7 the lowest.

The interrupt section jumper pad was connected in the following way:

(a) Interrupt Ø (pin 24) with (pin 34).

(b) interrupt 1 (pin 25) unconnected.

(c) Interrupt 2 (pin 26) with (pin 35).

(d) Interrupt 3 (pin 27) with (pin 41)

(e) Interrupt 4 (pin 28) with (pin 40).

(f) Interrupt 5 (pin 29) with (pin 63).

(g) Interrupt 6 (pin 30) with (pin 49).

(h) Interrupt 7 Unused.

3. Interrupt Handler routines.

Interrupt Ø was used to handle a special system event associated with Counter 1. Using system routine SETCDC(lapse,address) the user can request the system to execute the procedure whose address is indicated after the time lapse he gave. This system routine will set counter 1, which acts as a count down counter, with the specified time lapse. Counter 1, which is attached to interrupt Ø, will produce an interrupt when it gets to zero, and the interrupt handler will call the procedure as required. This mechanism must not be used frequently, nor should the procedure consume too much processor time.

Interrupt 1 was used to handle the monitor trap. The user can, by a command, activate the monitor when the processor gets to a certain execution point. To accomplish this, the monitor replaces the code at that point by an interrupt 1 command code. Whichever processor gets to that point first would execute the interrup 1 instruction and would then get to the MONITOR$TRAP procedure.

Interrupt 2 was used to update the real-time clock.

Interrupt 3 was used to schedule the serial input priority task. It is triggered by the Receiver Ready signal in the USART which goes high when a character is received from the keyboard.

Interrupt 4 was used to schedule the serial output priority task. It is triggered by the transmitter ready signal in the USART, which goes high when the USART buffer

has been emptied.

Interrupt 5 was used to schedule the parallel output priority task. It is triggered when a ready to receive signal is received from the printer.

Interrupt 6 was used for system reinitialization. It is connected to the front panel interrupt 6 switch.

D. COUNTERS.

1. Hardware Characteristics.

The SBC 80/20 includes an 8253 Programmable Interval Timer. The 8253 solves one of the most common problems in a micro-computer system, the generation of accurate time delays under software control. Instead of setting up timing loops in system software, the programmer configures the 8253 to match his requirements, initializes one of the counters of the 8253 with the desired quantity, then upon command the 8253 will count out the delay and interrupt the CPU when it has completed its task. It is easy to see that the software overhead is minimal and that multiple delays can easily be maintained.

Other counter/timer functions that are non-delay in nature but also common to most micro-computers can be implemented with the 8253.

- Programmable Baud Rate Generator
- Event Counter
- Binary Rate Multiplier

- Real Time Clock
- Programmable One-Shot
- Complex Motor Controller

The 8253 includes three separate counters. Each counter is a single 16-bit pre-settable down counter. Each counter can operate either in binary or in BCD and its outputs are configured by the selections of function stored in the Control Word Register and jumper configurations on the SBC 80/20.

The counters are fully independent and each can have separate mode configuration and counting operation, binary or BCD. Also, there are special features in the Control Word that handle the loading of the count value so that software overhead can be minimized for these fuctions.

The complete function definition of the 8253 is programmed by the system software. A set of control words must be send out by the CPU to initialize each counter of the 8253 with the desired function and quantity information. These control words program the function, loading sequence and selection of binary or BCD counting. Once programmed, the 8253 is ready to perform whatever timing task it is assigned to accomplish.

a. Mode 0: Interrupt on Terminal Count

The output of the 8253 counter will be initially low after the mode set operation. After the count is loaded into the selected count register, the output will remain low but the counter starts to count. When terminal count is

reached, the output will go high and remain high until the selected count register is reloaded, or the mode set again.

Reloading a counter during count will restart the procedure. A gate input will enable counting when high and inhibit counting when low.

b. Mode 2: Rate Generator

Divide by N counter. The output will be low for one period of the input clock. The period from one output pulse to the next is equal to the number of input counts in the count register. If the count register is reloaded between output pulses, the present period will not be affected, but the subsequent period will reflect the new value.

The Gate/Reset input, when low, will force the output high. When the Gate/Reset input goes high, the counter will start from the initial count. Thus, Gate/Reset input can be used to synchronize the counter.

2. The Real Time Clock

The Distributed System Real Time Clock is implemented using Counter 1 of the single board computer number one. The output from Counter 1 is attached to interrupt input pin number two and the count register loaded with a value that will produce and interrupt after one millisecond. The interrupt handler will increment the real time clock vector RTC and will reload the value into the count register to produce the next interrupt.

Counter 1, as all the counters in the 8253, counts at a rate of 930 nano-second. In order to produce an interrupt after one millisecond the count register must be loaded with a value close to 1000000ns / 930ns = 1075.2688. It was chosen 1075 = 433H.

SBC 1 is the only one whose interrupt mask is programmed to enable interrupt 2. It will update the real-time clock. Procedure INT2 in INTMSG accomplishes the update. The procedure checks for the variable, CLOCK, to be free before modifing the clock. This prevents it from disturbing any procedure which is reading the real time clock.

3. The Count Down Clock

The Count Down Clock is a feature implemented using Counter 0 from any SBC. This counter is programmed in mode 0, in the same way as Counter 1 for the real time clock. The user can request any processor to execute the indicated procedure as well as determine the count value. The system will load this value into the count register of Counter 0 and will enable interrupt 0, to which the counter is attached, by modifing the interrupt mask. The procedure entry point address is saved in variable CDCADR and a flag variable CDCACTIVE is implemented to ensure no side effect errors.

The user requests this service using system routine SETCDC(value,address) The value can not exeed FFFFH and the

user must remember that the rate is 930 nanoseconds. The procedure will be called by the interrupt handler INT0 in INTMSG.

4. Baud Rate Generation

Counter 2 has a dedicated function on the SBC 80/20. This counter provides a baud rate clock for to the serial I/O interface. The output from Counter 2 is made available to the 8251 USART where it can be jumpered to the Receiver and/or Transmitter clock inputs. Counter 2 is always enable (its gate input is tied to +5v).

Counter 2 must be programmed in mode 3 to generate the baud rate clock. As implemented, it generates a baud rate of 2400 bits per second.

E. SERIAL INPUT/OUTPUT INTERFACE

Each SBC would be able to handle a peripheral device connected to it by its J3 connector. This is a serial RS232 interface connector associated with the Universal Synchronous/Asynchronous Receiver/Transmiter (USART) communication interface. Receiver Ready (RxRDY) and Transmitter Ready (TxRDY) signal from the USART were wired to the interrupt controlled (interrupt 3 and 4) so the input/output functions would be performed by the priority task scheduled by the interrupt handlers 3 and 4. A set of procedures commanded from the system message task (number 0, 8 or 16) were implemented in order to monitor the operation

of these priority task.

1. Hardware Characteristics

The serial I/O interface logic provides the SBC 80/20 with a serial data communication channel that can be programmed to operate with most of the current serial data transmission protocols, synchronous or asynchronous. Baud rate, character length, number of stop bits and even/odd parity are program selected.

The serial I/O interface logic consist primary of an Intel 8251 USART device and several driver/receiver circuits.

Prior to starting data transmission or reception, the 8251 must be loaded with a set of control words generated by the CPU. These control words define the complete functional definition of the 8251 and must immediately follow a system reset operation (internal or external).

The control words are split into two formats:

- Mode Instruction
- Command Instruction.

Both the mode instructions and the command instruction must be given in a specified sequence for proper operation. The mode instruction must be inserted immediately following a reset operation, prior to using the 8251 for data communication.

All control words loaded into 8251 after the mode

instruction will load the command instruction. Command instructions can be writen into the 8251 at any time in the data block during the operation of the 8251. To return to the mode instruction format a bit in the command instruction word must be set to initiate an internal reset operation which automatically places the 8251 back into the Mode Instruction format.

a. Mode Instruction:

This format defines the general operational characteristic of the 8251. It must follow a reset operation. Once the mode has been written into the 8251 by the CPU, a SYNC character or command instructions can be inserted.

The 8251 can be used for either synchronous or asynchronous communication. The two least significant bits of the mode instruction control word specify the kind of operation.

b. Command Instruction:

Once the functional definition of the 8251 has been programmed by the mode instruction and the sync characters are loaded (if in sync mode), then the device is ready to be used for data communication. The command instruction controls the actual operation of the selected format. Functions such as: Enable Transmit/Receive, Error Reset and Modem controls are provided by the command instruction.

c. Data Transfers:

Once programmed, the 8251 is ready to perform its communication functions. The TxRDY output is raised 'high' to signal the CPU that the 8251 is ready to receive a character. This output (TxRDY) is reset automatically when the CPU writes a character into the 8251. On the other hand, the 8251 receives serial data from the modem or I/O device; upon receipt of an entire character, the RxRDY output is raised "high" to signal the CPU that the 8251 has a complete character ready for the CPU to fetch. RxRDY is reset automatically after the CPU read operation.

The 8251 cannot begin transmission until the TxEN (Transmitter Enable) bit is set in the command instruction and it has received a clear to send (CTS) input. The TxD output will be held in the marking state upon reset.

2. Serial I/O Priority Task

The actual character I/O functions are performed by a priority task, scheduled by the interrupt handler. Procedure CSINPUT in SEIO is the serial input priority task and CSOUTPUT in SEIO is the serial output priority task; they were defined as such in procedure SEIO$START in SEIO during system initialization.

CSINPUT puts the incomming characters into the serial input buffer CRTIN and also into the serial output buffer CRTOUT for echoing. Upon receipt of a carriage return character (CR), CSINPUT checks if the last string was requested by the connected module, in which case it will

sent the character string to the requesting module.

CSINPUT also identifies a special character defined by MONITOR$ID; when this character is sensed, the system routine MONITOR will be called which will activate the System Monitor.

CSOUTPUT dumps the serial output buffer CRTOUT one character at a time. It also senses if the connected module wants an acknowledge message when the buffer is empty, in which case CSOUTPUT sends the acknowledge message.

3. Serial I/O Monitoring Procedures

All requests for the use of the serial I/O priority tasks are sent to the System Message Tasks. The requests will be directed to message task 0, for computer 1; message task 8, for computer 2; and message task 16, for computer 3. These message tasks controls a set of procedures that keep track of the module connected to the facility.

Before any input string is passed, the connected task requesting for it is checked. Only a connected task can deposit data into the serial output buffer. The priority task CSOUTPUT would not be scheduled if the TxEN (Transmit Enable) bit in the USART command word were not set, and only the connected task has access to it.

4. USART Programming

The USART is programmed in procedure INIT$USART, called from SEIO$START in SEIO during system initialization.

It is set for asynchronous communication at 2400 bits per second. Formating details can be found in Reference [5].

F. PARALLEL OUTPUT INTERFASE

The distributed system provides also for parallel output devices to be connected to the single board computer. It makes full use of the hardware facilities and programmable device characteristics of the SBC 80/20. The acknowledge signal coming from the peripheral device after it accepts a sent character has been directed to the interrupt controller and connected to interrupt input pin number 5. In this way, once one character from the output string has been acknowledge, the subsequent priority task for sending the remaining characters in the string will be scheduled by the interrupt handler number 5.

1. Hardware Characteristics

The parallel I/O interface logic in the SBC 80/20 provides forty-eight (48) signal lines for the transfer and the control of data to or from peripheral devices. Sixteen lines have a bidirectional driver and termination networks permanently installed. The remaining thirty two lines are uncommitted. Sockets are provided for the installation of active driver networks or passive termination networks. The optional drivers and terminators are installed in groups of four by insertion into the 14-pin sockets.

All forty eight signal lines emanate from the I/O

ports on two Intel 8255 Programmable Peripheral Interface devices. The two 8255 devices allow for a variety of I/O configurations.

a. Operational Summary

The 8255 contains three 8-bit ports (A, B, and C). Each Port can be configured in a wide variety of functional forms by the system's software.

(1) Port A: One 8-bit data output latch/buffer and one 8-bit data input latch.

(2) Port B: One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

(3) Port C: One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under mode control. Each 4-bit nibble contains a 4-bit latch and it can be used for control signal outputs and status signal inputs in conjunction with Ports A and B.

The 8080 CPU controls the operating characteristics of the ports by sending two different types of control words to the 8255:

1) mode definition control word

2) Port C bit set/reset control word

Bit seven of each control word specified its format.

b. Mode Selection

There are three basic modes of operation that can be selected by the system software:

Mode Ø - Basic Input/Output

Mode 1 - Strobed Input/Output

Mode 2 - Bi-directional Bus

When the RESET input goes 'high', all ports will be set to Input mode Ø (i.e. all 24 lines will be in the high impedance state). After the RESET is removed the 8255 can remain in the input mode with no additional initialization required. During the execution of the system program, the other modes can be selected using a single output instruction. This allows a single 8255 to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed.

c. Single Bit Set/Reset Feature

Any of the bits of Port C can be set or reset with one output instruction. This feature reduces software requirements in control based applications.

When Port C is being used as status/control for Port A or B, each of its bits can be set or reset by using the Bit Set/Reset operation.

d. Interrupt Control Features

When the 8255 is programmed to operate in Mode 1 or Mode 2, control signals are provided that can be used as

interrupt request inputs to the CPU. The interrupt request signals, generated from Port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop. This function allows the programmer to disallow or allow specific I/O devices to interrupt the CPU without effecting any other device in the interrupt structure.

e. Mode 1

This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or 'handshaking' signals. In Mode 1, Port A and Port B use the lines on Port C to generate or accept these "handshaking" signals.

Mode 1 Basic Functional Definitions:

- Two transfer ports (A and B)

- Each transfer port contains one 8-bit data port and 4 bits from one half of the control/data port (Port C).

- The 8-bit data port can be either input or output. Both inputs and outputs are latched.

f. Port A

Port A is the most versatile of the three ports. It can be programmed to function in any of the three 8255 operation modes. This first port is the only port in each group that already includes a permanent bidirectional driver/terminator network, 8226 bus driver device at A1 and A2 (group one).

Before Port A is programmed for input or for

output in any of the three operation modes, certain jumper connections must be made to allow the port to function properly in the chosen mode. The 51-52-53 (group 1) jumper pad specifies the direction of data flow for the 8226 bidirectional bus drivers. If output in mode 1 is to be used, jumper pair 52-53 (group 1) should be connected.

When Port A is programmed for Mode 1, interrupts can be used. The INTR output from bit 3 of Port C activates the peripheral I/O interrupt request, PIA1. PIA1 is forwarded to the interrupt logic.

Because the 8226 bus drivers are inverting devices, all data is considered to be negative true with respect to the levels at the J1 or J2 edge connector.

g. Port C

As was described before, the use of Port C depends on the modes programmed for Port A and B. If Port A is in mode 1, bits 3, 4, 5, 6 and 7 of Port C can have the following dedicated control functions.

Bit 3 - INTR (Interrupt Request)

Bits 4 and 5 - Either input or output.

Bit 6 - ACK/ (acknowledge input)

Bit 7 - OBF/ (output buffer full)

2. Parallel Output Priority task

The Parallel output function is performed by a priority task, scheduled by th interrupt handler. The procedure LP$OUTPUT in PAIO is the priority task, defined as

such by procedure PAIO$START at system initialization. LP$OUTPUT dumps the parallel output buffer one character at a time. When the buffer is empty, it checks if the connected module wishes an acknowledge message to confirm the end of the output process, in which case it sends the message.

The process of outputting a character includes: setting the interrupt receiver for the acknowledge, sending the inverted character to the device, and then sending a strobe using bit 4 in Port C.

3. Parallel Output Monitoring Procedures

The Parallel Output priority task is monitored by a set of procedures. These procedures are controlled by the System Message Task, which receives the message requesting service from the task. In the same way as with the serial interface, the priority task is controlled by controlling the buffer, the parallel output buffer in this case. There is no command word controlling the hardware but the first character must be sent by calling procedure LP$STARTOUT.

4. Peripheral Interface Device

The Intel 8255 Programmable Peripheral Interface Device is programmed for using Port A for output in Mode 1. The proper hardware modifications as described in Reference [5] were done and the Terminator network and driver network installed in sokets A3 and A4 respectively.

G. THE SYSTEM MONITOR

The System Monitor was constructed with the idea of using it as a tool during the debugging of the applications programs. It has been built from a set of commands which will allow the user to examine, substitute, fill or move memory sections in common or on-board memory. It also has commands for changing the allocation of the system printer, for transmitting messages and for changing the allocation of the monitor itself.That is, for passing the connection with the system console from the monitor in one computer to the monitor in another.

There are two ways for setting up communications with the Monitor. One is by pressing the special key, identified by MONITOR$ID on the system console. The other is using the monitor command GO. If an address is specified following the command G, an interrupt 1 code will be inserted at that address. When the processor gets to this address point (if ever), the interrupt handler will call a procedure that will schedule the monitor to be executed.

All the Monitor procedures are attached to the system message tasks, the same as with the serial I/O and parallel output.

The Monitor commands are:

- D <address 1>,<address 2> : Dumps memory space from address 1 to address 2.

- F <address 1>,<address 2>,<character> : Fills memory locations from address 1 to address 2 with the character

specified.

- M <address 1>,<address 2>,<address 3> : Move memory block bounded by address 1 and address 2 to locations beginning with address 3.

- S <address> : Substitute subsequent memory locations beginning with location pointed by address. The sequence will finish with a carriage return.

- P <number> : Change system printer allocation to computer defined by number. Number must be between 1 and 3.

- T <number 1>,<number 2> : Transmit message <number 2> to message task <number 1>. If transmition succeeds the connection with the system consol is broken.

- C <number> : Change control to the computer indicated by number. Number must be between 1 and 3.

- G [<address>] : Break connection with the system console. Optionally, before leaving, the monitor could put a trap at location <address> for getting back to the system console when the processor gets to that point.

Observe that the Monitor is not active when it is not connected to the system console.

E. SYSTEM'S ADDRESS SPACE

All the program development for both the distributed operating system and the emulator was done using the system's programming language PLM80 under the ISIS-II operating system. The ISIS-II function LINK was used to link the program modules. All the external references that one

module would make were resolved by the LINK function. Function LOCATE, also from ISIS-II, was used to allocate the different program segments into memory. Using this function, all the memory references are resolved and the code is ready to be loaded. The command used was:

LOCATE DISIS.LNK CODE(0000H) MEMORY(C000H) STACK(3040H) DATA(31D2H) STACKSIZE(100H)

The programs were compiled separately for each computer, because of the difference in the system parameter CPTR, that identifies the host computer and the system parameter FIRSTMN, that identifies the number of the system message tasks related to that computer.

The standard SBC 80/20 was changed to increase the on board read only memory capacity. The necessary changes can be found in Reference [5] page 4-3.

I. PROGRAM DESCRIPTION

The Distributed System was divided into fourteen (14) program modules. Each module was placed in a different file. A file may contain one or more public procedures, or may contain public data declarations. Each file was compiled independently. Later, object programs generated by the compiler were linked together. The function of each procedure was described by the comment statement preceding it. All program listings were generated by the PLM/80 compiler.

1. File EXEC

File EXEC contains the program module EXECUTIVE which contains the procedure EXEC and the main program entry.

Procedure EXEC is the Distributed System Executive. This procedure manages the scheduled tasks and is the kernel of the system. As soon as EXEC gains control from the main program body, it calls procedure EXSTART, which initializes the system environment. Then EXEC will continue in a never-ending loop which form the heart of the system. There are three major sections in this loop: the priority task section, the message task section, and the periodic task section.

The priority task section carries out the tasks based on the variable PRIORSCHEDULE. This byte variable works as a vector flag. When a priority tasks needs to be served, the system routine PRIORITY will set a predetermined bit in this variable to one. There are eight priority tasks corresponding to the eight bits in this variable.

When the priority section finds that PRIORSCHEDULE is different from zero, it will look in the address vector PRIORLIST and extract the entry point address of the priority task and execute the task. When the priority task ends the flag bit will be reset to zero.

The message task section works based on the variable NUMMSG and EXTMSG. NUMMSG counts the number of messages in the buffer MSGBUFFER waiting to be processed. The header of

the message contains the destination message task number in the first byte. With this number the EXEC looks into the address vector MSGMODADDRESS for the task entry point and then executes it. In case the received message task is not between the eight assigned values of the executive, the message will be sent to the system buffer using system routine SENEXT. EXTMSG flags the presence of a message for a local message task in the system buffer. When the flag is set to the computer identification number of that single board computer, the executive will move the message to its buffer using system routine RECEXT.

The periodic message section works based on the variable NUMPER. This variable indicates the number of activated peiodic tasks. When the number is greater than Ø, the executive will examine each activated task and will check if the task activation time has arrived. It does so using system routines COPY$CLOCK and TIME$CMP. If the time has arrived, EXEC will pick up the entry point address from the address vector PERLIST and will execute the task. At the end of the task a new activation time will be computed using system routine SETPERTIME.

Priority tasks are checked first and immediately after any other task has been served. In this way message tasks will be served only if there is no priority task scheduled. Periodic tasks will be served only if there is neither a priority task nor a message task scheduled. When there is no task scheduled at all, an external light will be

lighted as an indication that the processor is idle.

The main program function is to check for the initialization variables. LOADSBC is checked to determine the proper moment for internal initialization. START is checked to determine if the computer is required in the actual configuration. Besides this, the main program body saves the initial stackpointer value for a possible reinitialization and sets the computer identification variable CPTR$ID.

2. File INTMSG

The program module INTMSG includes procedures for the interrupt handlers, the system message procedure, and the computer initialization procedures.

The interrupt handler procedures include a procedure with the attribute INTERRUPT for each one of the six interrupts used by the system, and a procedure INTRESET for the reset of the hardware interrupt mechanism. This module was compiled with the compiler parameter NOINTVECTOR so the loader would not try to construct an interrupt vector in common memory when the module was tested.

Interrupt Ø checks variable CDCACTIVE before calling the required procedure whose entry point address was saved in CDCADR. It also resets counter 1 to mode Ø so it will stop counting and will wait until the next value is loaded.

Interrupt 1 just calls the monitor routine MONITOR$TRAP that will wake up the local monitor.

Interrupt 2 updates the system clock, which is only updated by computer 1. The variable CLOCK must be checked before modifying the real-time clock in order to ensure that no other processor is in the middle of reading the multibyte clock value Notice that the four element vector RTC overlays variables RTCØ,RTC1,RTC2,RTC3, so that both variables can be used to reference the real-time clock. This was necessary because the code produced for index computation by the compiler would destroy the flag used by the operator PLUS if the vector increment mechanism were chosen.

Interrupt 3, 4, and 5 schedule a priority task by calling the system routine PRIORITY with the proper parameter.

Interrupt 6 restores the stack pointer STACKPTR to its initial value SAVESTACKPTR, resets the initialization varible LOADSBC to 1, and transfers control to SYSSTART.

MSGENTØ is the system message task procedure. It controls the use of the procedures for the serial I/O interface, for the parallel output interface, and for the monitor. Basically it receives messages that require the use of some of the procedures. It cheks the validity of the message and then transfers control to the corresponding procedure.

The initialization procedures are controlled by procedure EXSTART. EXSTART calls to SETEXDATA for data initialization, SETEXMSG for system messages initialization, and SETEXINTE for interrupt vector

initialization. Then, it programs counters Ø and 1, and in case computer 1 is in charge, it sets counter Ø with a value that produces the first 1 millisecond interrupt for the real time clock update. Finally, it calls SEIO$START for serial I/O initialization, PAIO$START for parallel output initialization, outputs the initial interrupt mask, and sends the "ready" message to the system console using system routine PRINT$ASYNC.

3. File SEIO

The program module SEIOMOD contains the procedures for controlling the serial I/O communication. Procedures CSINPUT and CSOUTPUT are the serial input and output priority tasks respectively. The other procedures are called from the system message task MSGENTØ to initialize the hardware and monitor the software over which these procedures perform.

CSINPUT generates inputs from the USART, character by character. It checks for special control characters like: CONSCL$ID for changing the system console to the local computer by resetting its system messages; MONITOR$ID for disconnecting from any other task and connecting with the monitor; DELINPUT for deleting the last character received; and CR for detecting the end of the input string and transmitting it to the connected task in case it was requested. If no special character was detected, the input character is saved in the input buffer CRTIN and the output

buffer CRTOUT for later transmission and immediate echoing respectively. CSOUTPUT picks up characters from the output buffer CRTOUT and sends them to the USART. When the buffer is empty, it will acknowledge it to the connected task if required, and then, it will disable the Transmit Enable Signal in the USART status, in order to stop the next scheduling of the task. Procedure STARTOUT is used for triggering the output sequence by enabling the Transmit Enable Signal in the USART.

SEIOSTART initialises the hardware and enters the priority task. The priority tasks are initialized using the system's routine ENTERPRIOR. The returned indices, used by the corresponding interrupts, are saved in variables USART$IN and USART$OUT.

All other procedures are used for loading or unloading the I/O buffers in one way or another and for controlling the use of the priority task by only the connected task.

4. File PAIO

The program module PAIOMOD contains the priority task LP$OUTPUT, the initialization procedure PAIO$START, and other procedures for the control of the parallel output buffer LPOUTBUF. LP$OUTPUT dumps the parallel output buffer character by character. Optionally it can send an acknowledge message to the connected task when the buffer is empty.

PAIC$START initializes the data and introduces the priority task using system routine ENTERPRIOR. The returning index PRELL$OUT is later used by the corresponding interrupt handler.

5. File MONI

The program module MONITOR has the procedures to carry out the monitor commands and the connection with the user tasks.

Procedure MONI$INTER is the command input procedure. It gets the command data string from the data portion of the message sent by the user task to MSGENTØ. It uses system routine GET$CHAR to extract the command's first letter. Upon comparison with the possible command codes it decides whether to transfer control to the corresponding procedure or ignore the command because it is invalid. After a command has been carry out or refused, the procedure will send a message input request for the next command using procedure RECEIVE$NEXT$COMMAND. This procedure uses system procedures PRINT$SYNC to output the promt character '.', and the system procedure CON$INPUT$REQ for requesting an input message from the system console.

6. File SC1

Program module SCPUP1 has the system routines of the lowest level. They are directed to set and check working data. The system routines are: FILL, CLEARDATA, SHFT, BIT,

CLEARBIT, SETBIT, and LEGAL.

- FILL(address1,address2,character). This routine will fill out memory from location <address1> to location <address2> with the character <character>.

- CLEARDATA(address1,address2). This routine will fill out memory from location <address1> to location <address2> with a blank character.

- SHFT(bit-position). This routine will return an 8-bit word with a 1 in position <bit-position>.

- BIT(bit-position,byte). this routine will return TRUE if the bit <bit-position> in byte <byte> is set. FALSE otherwise.

- CLEARBIT(bit-position,byte). This routine sets bit <bit-position> in byte <byte> to zero.

- SETBIT(bit-position,byte). This routine sets bit <bit-position> in byte <byte> to one.

- LEGAL(byte1,byte2,code). This routine returns TRUE if the value in byte <byte1> is lower than the value in byte <byte2>. Otherwise, it returns FALSE and outputs and error with code <code>.

7. File SC2

The program module SCPUB2 contains the system routines related to the periodic tasks and the system clock. It includes the system routines: COPY$CLOCK, SETPERTIME, TIME$CMP, PERACT, PERCHG, and PERSUSP.

- COPY$CLOCK(base). This routine copies the real

time clock into a four byte vector based in <base>.

- SETPERTIME(index). This routine sets the next activation time for the periodic task indicated by <index>. To do this, it will add the actual time, copied from the real time clock, to the period of the task and then it will save this result into the task activation time vector PERLIST(<index>).PERTIME.

- TIME$CMP(base1,base2). This routine will return a TRUE if the four-byte vector based on <base1> is lower than the four-byte vector based in <base2>. FALSE otherwise.

- PERACT(address1,address2). This routine will define a periodic task by filling out information on the periodic task data structure PERLIST. PERLIST(index).PERADR will contain the procedure address given by <address1>, PERLIST(index).PERINTADR will contain the address of the four-byte vector with the period given by <address2>.

- PERCHG(index,address). This routine is used to change the period of the periodic task indicated by <index> to the period indicated at the vector at location <address>.

- PERSUSP(index). This routine is used to suspend a periodic task. To do so, it is sufficient to define the data element PERLIST(<index>).FREE as TRUE and rearrange the index in the list PERXTBL.

8. File SC3

The program module SCPUB3 contains the system routines related to the message tasks. It includes the

system routines: MSGOVERFLOW, PACKMCB, SEND, GET, RELEASE, SETEXTMSG, RECEXT, SENEXT, and ILLEGALMSG.

- MSGOVERFLOW. This routine outputs the error code Ø1 for message buffer overflow.

- PACKMCB(address). This routine packs a four-byte vector from location <address> into the local message buffer MSGBUFFER.

- SEND(address). This procedure sends a message to a message task. It checks if the message task of the receiver as well as the sender is active, puts the header, located at <address> into the buffer, sets the pointer ADRMSGDATA to the location following to the header in the message buffer and returns a TRUE if this sequence has succeeded. Otherwise, it returns a FALSE.

- GET(semaphore). This routine will get the <semaphore> for the computer.

- RELEASE(semaphore). This routine release the <semaphore> from the computer.

- SETEXTMSG. This routine sets the value of EXTMSG for the use of the system buffer EXTMSGBUFFER.

- RECEXT. This routine moves a message from the system buffer EXTMSGBUFFER to the local buffer MSGBUFFER.

- SENDEXT. This routine moves a message from the local buffer MSGBUFFER into the system buffer EXTMSGBUFFER.

- ILLEGALMSG. This routine displays the illegal message signal on the system console.

9. File SC4

The program module SCPUB4 contains the system routines related to the priority tasks and interrupts. It includes system routines: PRIORITY, ENTERPRIOR, REMPRIOR, SETCDC, ENTER$INT, and REM$INT.

- PRIORITY(index). This routine is used to schedule the priority task indicated by <index>.

- ENTERPRIOR(address). This routine is used to enter the procedure whose entry point is at <address> as a priority task. The routine will return an index corresponding to the priority task.

- REMPRIOR(index). This procedure is used to suspend the priority task indicated by <index>.

- SETCDC(2-word-value,address). This routine is used to define a procedure at <address> that will be executed after .930 x <two-word-value> micro-seconds.

- ENTER$INT(index). This routine is used to enable the interrupt indicated by <index>.

- REM$INT(index). This routine is used to disable the interrupt indicated by <index>.

10. File SC5

The program module SCPUB5 contains the system routines related to the system console and the system printer. It includes the system routines: PRINT$ASYNC, PRINT$SYNC, CON$INPUT$REQ, CON$LINK$REQ, PAR$LINK$REQ, CON$RELEASE, CON$NEW$LINE, CON$BEG$LINE, PRINT$LINKED,

PAR$RELEASE, PAGE, WRITE, and WRITE$LINKED.

PRINT$ASYNC(address,count). This routine sends to the system console <count> bytes from text at location <address>.

- PRINT$SYNC(address,count,index). This routine sends to the system console <count> bytes from text at location <address> and the sender <index> that must correspond to the connected task.

- CON$INPUT$REQ(index,flag). This routine request an input string message from the system console. The message should be sended to message task <index> and the screen rotated or not depending on <flag>. The receiver must be 'connected'.

- CON$LINK$REQ(index). This routine connects message task <index> with the system console.

- PAR$LINK$REQ(index). This routine connects message task <index> with the system printer.

- CON$RELEASE. This routine disconnects the system console.

- CONNEWLINE(index). This routine produces a new line on the system console. The caller must identify himself by <index>.

- CONBEGLINE(index). This routine produces a new page on the system console. The caller must identify himself by <index>.

- PRINT$LINKED(address, count, index, code). This routine displays <count> bytes from location <address> into

the system console. <index> must identify the connected task. The system console will send back a message with the data <code> in it.

- PAR$RELEASE. This routine disconnects the system printer.

- PAGE(index). This routine produces a new page on the system printer. The caller must identify himself by <index>.

- WRITE(address,count,index). This routine displays <count> bytes from location <address> on the system printer. <index> must identify the connected task.

- WRITE$LINKED(address, count, index, code). Same as WRITE, but the system printer will send back a message with the data <code> in it.

11. File SC6

The program module SCPUB6 contains system routines for I/O computations mainly. It includes system routines: ACTIVE, UPDSTAT, CONVASC, DEBLK, GETNUM, VECTOR$ADD, VECTOR$SUB, NUMOUT, and GETCHAR.

- ACTIVE(index). This routine will return TRUE if message task number <index> is active.

- UPDSTAT(index,status). This routine will update the status of message task number <index> to <status>.

- CONVASC(byte). This routine will convert a 2-digit hexadecimal number <byte> into 2 ASCII characters. The characters will be left in the 2-word variable A4, least

significant bit first.

- DEBLK. This routine will skip the leading blanks in an input string message.

- GETNUM. This routine transforms an ASCII string into an hexadecimal value. The hexadecimal value will be stored into variable A4. The routine reference is used as a flag: it returns a TRUE if the tranformation was possible, FALSE otherwise. The ASCII string is taken from the message data and it must not exceed the decimal value of sixty five thousand five handred thirty six (65536).

- VECTOR$ADD(address1, address2, address3). This routine adds the four-byte vector at <address1> to the four-byte vector at <address2> and puts the result at the four-byte vector at <address3>.

- VECTOR$SUB(address1, address2, address3). This procedure subtracts the four-byte vector at <address2> from the four-byte vector at <address1> and puts the result into the four-byte vector at <address3>.

- NUMOUT(value,base,lc,address,width). This routine converts the value <value>, currently in base <base>, into an ascii string of at most <width> characters, and stores the string at location <address>. The unused left spaces are filled with the character <lc>.

- GET$CHAR. This routine gets an ASCII character from the input string message.

12. File SC7

The program module SCPUB7 contains three procedures for three system routines: ENTERMSGMOD that defines a new message task; MONITOR, that request the connection with the monitor; and ERROR that outputs an error message on the system console.

13. File MODPRO

The File MODPRO contains the procedures ENTER$MOD and CLEAR$MOD that are offered to the user for message definition under ISIS-II operating system. This file is supposed to be included in the user programs by means of the INCLUDE directive of the ISIS-II operating system.

14. File RUN

This file contains the program RUN that sets the initialization parameters LOADSBC and START and puts the MDS CPU into HALT state. RUN uses ISIS-II routines for reading the number of the computers the user wants to trigger from the console. It also constructs an interrupt handler environment that only allows interrupt six to be served from the MDS. To return to the ISIS-II, the MDS must be rebooted. The interrupt handler resets the interrupt request to allow the user to use interrupt 6 again.

IV. TACTICAL SYSTEM EMULATOR

A. DESIGN

The Tactical System Emulator is based on the idea that a Tactical System can be partitioned into functional modules and that each of these functional modules may be replaced by dummy modules that will demand the same processing time and intermodule communication effort from the hardware.

Replacing the real tactical system by a dummy system will allow us to study the behaviour of the hardware, in this case a distributed micro-computer architecture, without programming each module. The system designer will only concern himself with the definition of the functional parts that constitute his tactical system and then exercise the emulation in order to study the proposed system's behaviour.

Two kinds of functional modules are recognized: Periodic functional modules and Demand Functional Modules. Periodic Functional Modules are activated periodically by the controlling mechanism and are usually in charge of updating data structures or pooling data from or to some peripheral. Demand functional modules are activated upon demand from another functional module, these are in charge of most of the computations.

1. Dummy Function

All functional modules will be replaced by dummy tasks (periodic or message) and data structures which

determine their performances. These tasks will be directed to consume an amount of processor time similar to the the estimated execution by the functional module they are replacing. Also they transmit the same messages, with the same destination and length as the modules which they replace.

The data block that contains the information on the basis of which the dummy task emulates the functional module is conceptualized as an activation record. All functional modules that are defined in the system will have one activation record. The dummy tasks do not need to be all different because they will perform based on different activation records. In fact, only two (2) dummy tasks are needed: one for the periodic functional modules and one for the demand functional modules.

All activation records include:

- Functional module identification number
- Processor time needed
- Number of message that are transmited
- Destination, Length and number of each message
- Total Time in Execution
- Number of times Activated

Besides this, periodic activation records contain:

- Period (in real-time clock format)

The demand activation record contains:

- Number of States Before Execution
- Number of input messages

- Message number, number of times received and total delay for each input message.

2. Emulation Sequence

There are three very definite steps in using the emulation system:

The first one, as already mentioned, is identifying the functional components of the tactical system.

The second one, is creating the dummy system by interacting with the dynamic tactical system definition tool. In this interaction the user defines the real system parameters to the emulator.

The third step is running and collecting statistics from the emulator. After the statistics have been collected, the user can go back to step 2 and modify the parameter or move the allocated dummy functions from one computer to another.

B. IMPLEMENTATION

Each computer will be able to handle up to eight periodic and eight demand dummy modules. The corresponding activation records were allocated in on board memory and the code in common memory. In this way all three computer will share the same code for the dummy modules and the activation record over which they perform will be different for each processor. The same is true with the code that dumps the statistics from the activation records.

1. Emulator-System Relation

The emulator uses message task three (3) for controlling the emulation operation. This message task will connect itself with the system console from which it will get the input message coming from the user. Message 3 will receive the real tactical system parameters together with the emulator environment the user wishes to use. Message 3 will send these parameters to message tasks in the other computers in order to make them construct the activation records.

Message task number 2, 10, and 18 in computers 1, 2, and 3 respectively were used to contruct the activation record. They receive the necessary information from message task 3. Message tasks 1, 9, and 17 are the the dummy demand modules for computers 1, 2, and 3 respectively. Observe that the three message tasks use the same code in common memory. The same is true for message tasks 2, 10, and 18.

Message tasks 4, 12, and 20 were used for dumping the data and statistics in the activation record. They become active upon receipt of a message from message task 3 after the user has requested the dump. They are also used to move the activation record from one computer memory to another, corresponding to moving a dummy module from one processor to another.

Message tasks 3, 4, 12, and 20 were defined in the main body of the emulator program. They will be executed under ISIS-II operating system so that the procedures used

for the definitions were from program MODPRO. Message tasks 1, 9, 17, 2, 10, and 18 were defined in procedure SET$MODS, executed upon receipt of the system initialization message by message number three. Procedure SET$MODS uses system routine ENTERMSGMOD for this purpose.

2. Emulation Control

The emulation is initiated from the system point of view by activating the periodic tasks that replace the periodic functional modules. They are supposed to trigger or initiate the message sequences that would activate the system.

Periodic tasks are activated by message task 2, 10, and 18 upon receipt of the commanding message from message task 3. They use system routine PERACT(Address,Period) for this purpose.

The emulation can be stopped in two ways: First, the user could define a 'sink' module that would count the number of times the same message has been sent to him, and when this count gets to a predetermined number it would send a message to the system that would suspend all periodic tasks and cancel all message task. Second, the user could specify a time limit, using the count down clock CDC. The emulator would count to this limit and when reached, it would produce the same stopping message as in the previous case.

3. Linking and Locating

The emulator programs must be linked together as well as the referenced system routines. Because system routines were already located in local Read Only Memory, the linking must be done with the PUBLIC references of the distributed system code. The linking command for the emulator was:

LINK :F1:EMULA1.OBJ, :F1:EMULA2.OBJ, :F1:EMULA3.OBJ, :F1:EMULA4.OBJ, SYSTEM.LIB(EXIT), SYSTEM.LIB(ISIS), PUBLICS(DISI), PLM80.LIB TO :F1:EMULA.LNK

The emulator code was located in common memory starting at 5000H to allow the code to be shared by more than one processor. The emulator data must be located between locations 3900H and 4000H. The command used was:

LOCATE :F1:EMULA.LNK CODE(5000H) DATA(3900H)

C. PROGRAM DESCRIPTION

1. File EMULA1

This program module contains PERIODIC$MOD and DEMAND$MOD, the periodic and demand dummy procedures, the ones that will be executed instead of any periodic or demand functional module that exist in the real tactical system. It also has DATA$FILLER, the procedure that is shared by message tasks 2, 10, and 18.

The procedure SEND$MSGS(Base) is used to send the messages between the dummy functions. It takes the

information from the activation records and only requires a base to the variable NUMMSGOUT.

The procedure EMULATE(Base) consume an amount of times dictated by the variable pointed to by <base> and increments the execution time and total executed time in the corresponding variable.

CORRECT$INPUT$MSG verifies that the message received is correct and updates the statistics about that message. It receives the coming message identification number and a base to the coming message list.

CORRECT$ID checks whether the scheduled function to be emulated is correct or not and sets the activation record pointer MODE$BLOCK.

Observe that in this program, the mnemonics DBD, DBP, DTA, and MBK have been used instead of the references DATA$BLOCK$DEM, DATA$BLOCK$PER, MSG$DATA, and MOD$BLOCK respectively. Procedures SEND$MSGS, EMULATE, and the CORRECT$INPUT$MSG are used by both the DEMAND$MOD and PERIODIC$MOD.

2. File EMULA2

This module contains the emulator program's main program, the emulator coordinator task, and procedures for the control of the principal functions of the emulator.

The main program body will be executed under ISIS-II operating system. It uses procedures CLEAR$MOD and ENTER$MOD to initialize the emulator message tasks and the ISIS-II

system routine EXIT to give the control back to the user after this initialization. Observe that the file :F1:MODPRO.SRC has been included as part of the module and that the routine EXIT has been declare EXTERNAL.

MSGENT3 serves as a coordinator for the emulator to establish the relation with the user through the system console and the line printer and it directs the steps of the emulation. Upon receipt of the initialization message (message number 0), it will call INIT$EMU. INIT$EMU initializes the other emulator message tasks by calling procedure SET$MODS, clears its local data using the system routine CLEAR$DATA, and request the use of the console by using the system routine CON$LINK$REQ. The system responds to the console with the message number 21. When MSGENT3 receives message 21 it calls procedure CHECKSECONNECTION. This procedure will keep asking from the system console if the previous request was denied or it will initiate the interaction with the user for setting up the parameters for the emulation. CKECKSECONNECTION asks the first response from the console by activating the routine CON$INPUT$REQ. The inputs from the console will came with message number 20. When this message number is received, MSGENT3 will call RECEIVE$INPUT, that belongs to programming module EMULA4.

When MSGENT3 receives the message number 30 it will stop the emulation by calling the procedure STOP$EMU. This procedure sends message to message task 2, 10, and 18 that will suspend the periodic tasks and it also will deactivate

the message task used as dummy functions. Besides this, it will also read the finishing time and compute the time taken by the emulator and put it into EMU$CLOCK$2.

EMULA2 also contains procedures for the control of the principal functions of the emulator, they include: Start Emulation (START$EMU), Move task (MOVE$TASK), Substitute data (SUBS$DATA), and Write data and statistics (WRITE$DATA).

START$EMU sends the triggering message for the activation of the periodic tasks (it initiates the emulation), and saves the emulation initialization time in variable EMU$CLOCK$1. In case a time limit emulation is chosen, START$EMU defines a priority task that will be called after the "count down clock" event. Procedure EMU$COUNTER will be a priority task and it will be scheduled every fifty milliseconds.

MOVE$TASK moves an activation record to another computer. It receives its data from the message data, the first byte of which should be the the activation record number that must be moved, and the second byte the receiving computer number. With these two bytes, and examining the data structures ID$TBL and MSG$TBL, MOVE$TASK constructs the appropriate message with the information for constructing the activation record in the receiving computer. The message is sent to message tasks 4, 12, or 20 (procedure MSGENT4).

SUBS$DATA substitutes previously sent data. It interprets the command and directs the user to the iteration

point at which the data was sent, so that the process is repeated.

WRITE$DATA interprets the writing command and then uses the procedure SEND$WRITE$MSGS to send the appropriate message to MSGENT4.

3. File EMULA3

The programming module EMULA3 includes the procedures for writing out the real system parameters and the collected statistics on the system console or the line printer. It also includes the procedure for reordering the activation record in local memory. Both functions are controlled by procedure MSGENT4, used by message tasks 4, 12, and 20.

MSGENT4 will set up a print flag vector variable called PRN$FLAG and then will call procedure SET$WRITER which in turn will establish connection with the system console or the line printer; this is based on the message number and the data byte received with it. When the connection is acknowledged with message number 28, procedure PRN will print out the desired data. PRN prints the data, sending messages in batches. The last message of a batch requests an acknowledge from the printing message task. The acknowledge message is message number 26. When MSGENT4 receives message number 26 it just calls procedure RECCOTBCODE which continues with the next batch.

MSGENT4 also receives the message that trigger

procedure SEND$TASK that sends a required activation record to a specified computer, REC$TASK that receives the activation record coming from another computer, and SETNEWSINK that changes the destination of all the messages directed to the dummy function that uses the moved activation record.

4. File EMULA4

The program module EMULA4 contains the procedures needed for the interaction with the user in order to receive the real time system and emulation parameters. All these procedures are directed by procedure RECEIVE$INPUT that is called by MSGENT3.

The program EMULA4 has several entry points. The entry point a particular activation gets to, depends on the state of the interaction with the user. This state is controlled by saving a value in variable STATE.

When variable STATE has the value Ø, the entry point will be procedure STATEØ; when variable STATE has the value 1, the entry point will be procedure STATE1; and so on. There are up to 32 possible values for variable STATE with its corresponding entry point procedures. The last entry point, corresponding to a value equal to 32 in variable STATE, is procedure STATE32.

The variable STATE is modified after a particular question has been responded by the user, or after a limit has been reached.

V. CONCLUSIONS

Multi-micros can be used by dividing a tactical application into explicit asynchronous (but cooperating) processes. The emulator makes it possible to evaluate alternative process assignments (to processors) as an aid in designing the process structure. Future work is required to evaluate how many processors can be effectively used.

There is a need for an application independent operating system for multi-micros. Much of this thesis effort was dedicated to tailor the special purpose operating system on hand. The operating system should manage global/local memory. A more efficient emulator could be constructed if the operating system provides a mechanism for traslating the dummy functions code between local and global memory.

There is a trade-off between using more memory for storing similar copies of a code in each computer local memory or increase the possibility of bus contention by sharing code in common memory.

The emulator design is not tied to whether or not the functional modules are in local or global memory, but this implementation links and locates programs in a way that the emulator only represents structures of multiple computers sharing code in common memory.

An emulator for structures which use common memory strictly for interprocess communication can be derived from the same programs by locating the module EMULA1 in on-board

memory. In order to locate EMULA1 in on-board memory, some modules from the O/S must be disregarded, because of the limited space. Modules PAIO and MONI are the most likely to be disregarded for this purpose.

APPENDIX A. SBC 80/20-4 DESCRIPTION

The SBC 80/20-4 is a member of Intel's complete line of OEM computer systems which take full advantage of Intel's LSI technology to provide economical, self contained computer based solution to OEM applications. The SBC is a complete computer system on a single 6.75-by-12 inch printed circuit card. The CPU, system clock, read/write memory, non-volatile read-only memory, I/O ports and drivers, serial communication interface, interval timer, interrupt controller, bus control logic and drivers all recide on the board.

To facilitate the following description, the SBC 80/20 can be devided into eight major functional blocks:

1) CPU Set
2) Bus Interface
3) Random Access Memory
4) Read Only Memory
5) Serial I/O interface
6) Parallel I/O interface
7) Interval Timer
8) Interrupt Controller

The CPU Set consist of the 8080A Control Processor, the 8224 Clock Generator and the 8238 System Controller. The CPU Set is the heart of the SBC 80/20. It performs all system processing functions and provides a stable timing reference for all other circuitry in the system. The CPU generates all

of the address and control signals necessary to access memory and I/O ports both on the SBC 80/20 and external to the SBC 80/20. The CPU set is capable of fetching and executing any of the 8080's instructions. The CPU set responds to interrupt requests originating both on and off the SBC 80/20, and to WAIT requests from memory or I/O devices having an access time which is slower than that required by the SBC 80/20's 8080A cycle time.

The Bus Interface allows the SBC 80/20 to use a common system bus with other master devices such as other CPUs or DMA devices, thus sharing common memory and I/O resources. The Bus Interface includes an Intel Bus Controller, as well as circuits for the generation of the Bus Clock signal (BLCK/). The Bus Controller arbitrates all SBC 80/20 request for use of the system bus, synchronously with respect to the Bus Clock. When the SBC 80/20 acquires control of the Bus, The Bus Controller generates the appropiate memory or I/O command signal, gates the address into the system address lines and gates data on/off the system bus. Operation between the CPU and on board resources require no use of the system bus.

The Random Access Memory (RAM) section provides the user with 4096 (2K) x 8-bits of read/write storage, using eight Intel 2114 static RAM devices. The 2114 requires neither refreshing nor clock input to operate. All operations between the CPU and on board RAM require no WAIT states.

The Read Only Memory (ROM) section provides the user

with the necessary provisions for installing 4096 x 8-bit or 8192 x 8-bits of ROM or EPROM. Each SBC 80/20 has four 24-pin sockets that can accept either Intel 8708 Erasable and Electrically Programmable Read Only Memory (EPROM) chips (1024x8-bits each) or Intel 8308 static MOS Read Only Memory chips. Each SBC 80/20 also includes the option of installing four 2048 x 8-bits Intel 2716 Erasable and Electrically Programmable Read Only Memory Chips. The board includes the necessary acknowledge and address decoding circuitry. All CPU accesses to this on-board ROM/EPROM area require no CPU WAIT states.

The serial I/O interface, using Intel's 8251 USART device, provides a full duplex RS232 serial data communication channel that can be programmed to operate with most of the current serial data transmission protocols. Synchronous or asynchronous mode, baud rate, character length, number of stop bits and the choice of even, odd or no parity are all program selectable.

The Parallel I/O Interface, using the Intel's 8255 Programmable Peripheral Interface device, Provides 48 signal lines for the transfer and control of data to or from peripheral devices. Sixteen lines already have a bi-directional driver and termination network permanently installed. This bi-directional network allows these sixteen lines to be inputs, outputs, or bi-directional (selected via jumpers). The remaining 32 lines, however, are uncommitted. Sockets are provided for the installation of driver or

terminator networks as required to meet the specific needs of the user system.

The Interval Timer capability is implemented with an Intel 8253 Programmable Interval Timer. The 8253 includes three 16-bit BCD or binary counters which can be programmed by the user to perform a variety of timing functions. The output from the first two counters can be used as interrupt request lines, thus allowing simple implementation of such features as a real time clock or a program monitor alarm. The output from the third counter is applied to the Serial I/O interface. When properly programmed, this counter can provide the desired baud rate frequency for serial communications.

The SBC 80/20 also includes an Intel 8259 Interrupt Controller. The 8259 device resolves priority among eight different interrupt levels according to an algorithm that is programmed by the user. A jumper area in the interrupt section permits the user to connect any of nine external bus interrupt lines or 17 on-board interrupt request to eight priority levels inputs to the 8259. Thus, by jumpering various interrupt lines to the appropiate priority level and by programming the 8259 with the desired algorithm, the user can easily configure a custom interrupt structure.

APPENDIX B. MULTIBUS DESCRIPTION

A significant measure of the INTELLEC MDC System's power and flexibility can be attributed to the design of its bus. The bus structure allows for multiple master-slave relationships between the various system modules. In fact, the bus can support eight masters in a parallel, priority network. By connecting adjacent masters modules with a serial bus priority line, the maximum number of masters can be expanded to 16. In such a configuration, each master pair contends for bus control, the two masters in the pair further resolve contention via the serial priority line. This configuration allows for an increased number of master modules without incurring the timing overhead of a pure serial network. Where a pure serial bus control network is implemented on the INTELLECT MDS Bus, the maximum bus transfer rate of 5 MHz can't be guaranteed in that application.

The Bus provides its own clock which is derived independently from the processor clock. The Bus clock provides a timing reference for resolving bus contention among multiple bus requests. This feature allows different speed processors to share resources on the same bus. Actual transfer via the bus, however, proceeds asynchronously with respect to the bus clock. Thus, the transfer speed is dependent on the transmitting and receiving devices only. This design prevents slow master modules from being

handicapped in their attepts to gain control of the bus, but does not restrict the speed at which faster modules can transfer data via the same bus. Once a bus request is granted, single or multiple read/write transfers can proceed at a maximum rate of 5 MHz. The most obvious application for the master-slave capabilities of the bus is multi-processor configurations and high speed direct-memory-access (DMA) operations, but are by no means limited to these two.

The INTELLEC MDS System Bus (excluding power inputs) consist of 56 signal lines, including 16 address lines, 16 bidirectional data lines, and 8 multi-level interrupt lines. Thus, the system is capable of supporting 64K (65,536) words of storage.

APPENDIX C. HOW TO USE THE EMULATOR

The Tactical System Emulator is a tool designed to help the user in the allocation of tactical system fuctional modules into a distributed micro-computer architecture.

A Tactical System is viewed as a set of functional modules that interact with each other seeking the same goal. Two functional module types are recognized: Periodic and Demand. Periodics will be activated by the system in a constant period and Demands will be activated upon receive of a message with data input.

The emulation sequence consist of three steps:

(a) the definition of a network that characterizes the process structure of the tactical system,

(b) the definition of the network parameters to the emulation system,

(c) the control of the emulation through emulation commands.

A. NETWORK DEFINITION

Every Tactical System can be partitioned into functional modules that are executed as separate processes. This package does not support the software for helping in the definition of a tactical system network, but several methods can be found in References [3] and [4].

The emulator supports up to eight periodic modules and eight demand modules in each computer. Each module can have

up to four sending message numbers, each message number with a single destination. Each demand module can have up to four receiving message numbers.

B. PARAMETER DEFINITION

The emulation parameters are requested and checked by the emulator system itself. Upon system initialization the emulator will be switched automatically to the input state where the user will specify the number of computers needed and the modules allocated in each computer with its corresponding parameters.

The initialization sequence is as follows:

- Bootstrap ISIS-II operating system in MDS Micro-computer Development System. The hardware should include the standard boards with 64K of memory and the required number of Single Board Computers connected. Each SBC with the corresponding copy of the Distributed System on ROM.

- Run the program EMULA. (EMULA contains the emulator code)

- Run the program RUN with the number of the SBCs you want to use. (e.i." RUN 1 2 " will authorize SBC 1 and SBC 2).

- The MDS console will be in HALT, and a the emulator identification tag will appear in the Distributed System Console.

- From there on, you must respond to the emulator

requests in order to define the network.

C. EMULATION CONTROL

After the network parameters have been defined the emulator will put itself into a wait state and will display the possible commands it is able to execute. They include:

W <H><P/D/S><1/2/3> - write data

E - emulate

S <#/1/2/3/P> - substitute data

M #<,1/,2/,3> - move task number #

"W" is a request for displaying data. If only "W" is keyed, all the periodic module, demand module, and statistic data from computers 1, 2, and 3 will be displayed in the system console. If letter "H" is specified the output will be directed to the line printer. P, D, or S, specify a subset to the data corresponding to periodic, demand or statistics data. 1, 2, or 3 specifies a subset of the data corresponding to computers 1, 2, or 3.

"E" initiates the emulation. At the end of the emulation, the same commands will be available.

"S" is used to change some of the network parameters. If it is keyed alone, the input process will be repeated from the beginning. Otherwise, a specified module number #, computer 1, 2, or 3, or the emulation parameter (Time limit or not) data will be changed.

"M" moves module number # to computer 1, 2, or 3.

DISTRIBUTED SYSTEM PROGRAM LISTING

The following table summarizes the content of the distributed system files.

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE RUN
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:RUN.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)
```

```
                /**********************************************************
                ***********************************************************
                **
                : THIS PROGRAMS READS THE NUMBERS OF THE SBCS NEEDED
                AND SETS PARAMETERS LOADSBC AND START FOR TRIGGERING
                THEM. IT ALSO PROVIDES WITH AN INTERRUPT RESET ROUTINE
                FOR MDS INTERRUPT NUMBER SIX.
                **
                ***********************************************************/
  1             RUN: DO;
  2    1           DECLARE DI LITERALLY 'ØF3H';
  3    1           DECLARE LOADSBC BYTE AT(ØF65CH);
  4    1           DECLARE START(3) BYTE AT(ØF65DH);
  5    1           DECLARE INBUF(5) BYTE;
  6    1           DECLARE COUNT BYTE,
                       ACTUAL BYTE,
                           STATUS BYTE;

  7    1           READ: PROCEDURE(P1,P2,P3,P4,P5) EXTERNAL;
  8    2             DECLARE(P1,P2,P3,P4,P5) ADDRESS;
  9    2             END READ;

 10    1           INT6: PROCEDURE INTERRUPT 6;

 11    2            OUTPUT(ØFDH) = 2ØH; /* RESTORE INTERRUPT LOGIC */
 12    2            HALT;
 13    2            END;


 14    1             OUTPUT(ØFDH) = 12H; /* RESET INTERRUPT LOGIC */
```

```
 15   1            OUTPUT(0FCH) = 0 ;/* DITTO */
 16   1            OUTPUT(0FCH) = 10111101B; /* INT. 6 AND 1 ALLOWED */

                  /*************************************************
                        READ PARAMETERS AND SET VARIABLES
                     *************************************************/
 17   1            CALL READ(1,.INBUF,5,.ACTUAL,.STATUS);
 18   1            COUNT = 0 ;
 19   1            DO WHILE ACTUAL <> 0 AND COUNT < 5 ;
 20   2                IF INBUF(COUNT) = '1' THEN START(0) = 1;
 22   2                IF INBUF(COUNT) = '2' THEN START(1) = 2;
 24   2                IF INBUF(COUNT) = '3' THEN START(2) = 3;
 26   2                ACTUAL = ACTUAL - 1;
 27   2                COUNT = COUNT + 1;
 28   2                END;
 29   1             LOADSBC = 1; /* TRIGGER SBC 1 */
 30   1             HALT; /* HALT COMPUTER */
 31   1       END;
```

97

MODULE INFORMATION:

```
     CODE AREA SIZE      = 009DH    157D
     VARIABLE AREA SIZE  = 0008H      8D
     MAXIMUM STACK SIZE  = 0008H      8D
     48 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE EXECUTIVE
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:EXEC.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            EXECUTIVE:   DO;
                $ INCLUDE(:F1:SYDATP.SRC)
            =   /*
            =   ************************************************************************
            =   ************************************************************************
            =   **
            =                       SYSTEM DATA DECLARATIONS
            =   ***
            =   */
   2    1   =   DECLARE LIT LITERALLY 'LITERALLY',
            =           DCL LIT 'DECLARE';
            =
   3    1   =   DCL TRUE LIT '0FFH',
            =       FALSE LIT '0',
            =       RMN LIT '0',
            =       SMN LIT '1',
            =       MN LIT '2',
            =       ML LIT '3',
            =       MAXCPTR LIT '3',          /* MAX NUMBER OF COMPUTERS */
            =       MAXMOD LIT '8',           /* MAX NUMBER OF MODULES IN 1 CPTR */
            =       MAXSYSMOD LIT '24',       /* MAX NUMBER OF MODULES IN SYSTEM */
            =       MSGBUFLEN LIT '512',      /* LENGTH OF MSG BUFFER */
            =       MAXPRIOR LIT '8',         /* MAX NUMBER OF PRIORITY CALLS */
            =       MAXPER LIT '8',           /* MAX NUMBER OF PERIODIC CALLS */
            =       BEGINCM LIT '0F400H',     /* BEGIN OF COMMON MEMORY */
            =       INTVECTOR LIT '03000H',   /* INTERRUPT VECTOR */
            =       CPTR LIT '1',             /* NUMBER OF HOST COMPUTER */
            =       FIRSTMN LIT '0',          /* MN OF FIRST MODULE IN CPTR */
            =       LASTMN LIT 'FIRSTMN+7',   /* MN OF LAST MODULE IN CPTR */
            =       EX   LIT 'FIRSTMN',       /* MN OF EXEC MODULE */
```

```
          =       MSG$TBL$ADR LIT '0F660H';/* MSG. ENTRY TABLE ADDRESS */
          =
          =   /*
          =   ***
          =   */
          =
4    1    =   DCL (B1,B2,B3) BYTE PUBLIC,
          =       (A1,A2,A3,A4) ADDRESS PUBLIC,
          =       (BL,BU) BYTE PUBLIC AT (.A4);
          =             /*
          =                   SYSTEM WORK VARIABLES
          =                   BU(UPPER) AND BL(LOWER) OVERLAY ADDRESS VARIABLE A4
          =                   */
5    1    =   DCL (ADRMSG,ADRMSGDATA) ADDRESS PUBLIC,
          =       (MSG BASED ADRMSG) (4) BYTE,
          =       (MSGDATA BASED ADRMSGDATA) (100) BYTE;
          =             /*
          =                   WORK AREAS FOR MSG CONTROL BLOCK (MSG) AND MSG
          =                   DATA BYTES (MSG DATA)
          =                   ADRMSG AND ADRMSGDATA ARE SET BY EXEC BEFORE THE
          =                   CALL OF A MODULES MSG HANDLER
          =                   */
6    1    =   DCL INTMASK BYTE PUBLIC;
          =               /*
          =                   MASK OF CURRENTLY ACTIVATED INTERRUPTS
          =                   */
7    1    =   DCL CPTR$ID BYTE PUBLIC ;
          =
```

```
                $EJECT
                $ INCLUDE(:F1:EXDATP.SRC)
            =   /*
            =   ***************************************************************
            =   ***************************************************************
            =   **
            =                               EXECUTIVE DATA
            =   ***
            =   */
            =
  8     1   =   DCL INT0$LOC BYTE PUBLIC AT (INTVECTOR),
            =       INT1$LOC BYTE PUBLIC AT (INTVECTOR +  8),
            =       INT2$LOC BYTE PUBLIC AT (INTVECTOR + 16),
            =       INT3$LOC BYTE PUBLIC AT (INTVECTOR + 24),
            =       INT4$LOC BYTE PUBLIC AT (INTVECTOR + 32),
            =       INT5$LOC BYTE PUBLIC AT (INTVECTOR + 40),
            =       INT6$LOC BYTE PUBLIC AT (INTVECTOR + 48),
            =       INT7$LOC BYTE PUBLIC AT (INTVECTOR + 56),
            =       INT0$PRO ADDRESS PUBLIC AT (INTVECTOR +  1),
            =       INT1$PRO ADDRESS PUBLIC AT (INTVECTOR +  9),
            =       INT2$PRO ADDRESS PUBLIC AT (INTVECTOR + 17),
            =       INT3$PRO ADDRESS PUBLIC AT (INTVECTOR + 25),
            =       INT4$PRO ADDRESS PUBLIC AT (INTVECTOR + 33),
            =       INT5$PRO ADDRESS PUBLIC AT (INTVECTOR + 41),
            =       INT6$PRO ADDRESS PUBLIC AT (INTVECTOR + 49),
            =       INT7$PRO ADDRESS PUBLIC AT (INTVECTOR + 57);
            =
  9     1   =   DCL MSG$MOD$ADDRESS(MAXSYSMOD) ADDRESS PUBLIC AT (MSG$TBL$ADR);
            =                       /* ADDRESSES OF MODULES'S MESSAGES ENTRY POINTS */
            =
 10     1   =   DCL MODSTATUS (MAXSYSMOD) BYTE PUBLIC AT (BEGINCM);
            =           /*
            =           MODSTATUS CONTAINS THE STATUS OF ALL MODULES
            =           IN THE SYSTEM.INDEX IS MODULE NUMBER.
            =           (BIT 0 = LSB)
            =           BIT 0 - BIT 7 = 0: MODULE DOES NOT EXIST
```

```
=           BIT 0: 1 - MODULE EXISTS
=           BIT 1: 0 - MODULE NOT ACTIVATED
=                  1 - MODULE ACTIVATED
=           BIT 2: 0 - MODULE MAY NOT BE SUSPENDED
=                  1 - MODULE MAY BE SUSPENDED
=           BIT 3: 0 - MODULE NOT AUTHORIZED TO SUSPEND/
=                     ACTIVATE OTHER MODULES
=                  1 - MODULE AUTHORIZED TO SUSPEND/ACTIVATE
=                     OTHER MODULES
=           */
11   1  =  DCL ADRSTAT ADDRESS PUBLIC,
=           (MODSTAT BASED ADRSTAT) (MAXMOD) BYTE;
=           /*
=           MODSTAT IS AN OVERLAY FOR MODSTATUS AND COVERS
=           THE ENTRIES FOR ONE COMPUTER.
=           THE BASE VARIABLE ADRSTAT IS SET AT START.
=           */
12   1  =  DCL BEGABSDATA BYTE PUBLIC AT (BEGINCM+MAXSYSMOD),
=           /*
=           BEGIN OF ABSOLUTE DATA IN COMMON MEMORY
=           */
=           (RTC0,RTC1,RTC2,RTC3) BYTE PUBLIC AT (.BEGABSDATA),
=           RTC (4) BYTE PUBLIC AT (.BEGABSDATA),
=           CLOCK BYTE PUBLIC AT (.BEGABSDATA + 4),
=           /*
=           SYSTEM REALTIME CLOCK
=           LEAST SIGNIFICANT BYTE = RTC(3)
=           LEAST SIGNIFICANT BIT : 1 MSEC
=           */
=           SYSMN10(4) BYTE PUBLIC AT (.BEGABSDATA + 5),
=           SYSMN11(4) BYTE PUBLIC AT (.BEGABSDATA + 9),
=           SYSMN12(4) BYTE PUBLIC AT (.BEGABSDATA + 13),
=           SYSMN13(4) BYTE PUBLIC AT (.BEGABSDATA + 17),
=           SYSMN14(4) BYTE PUBLIC AT (.BEGABSDATA + 21),
=           SYSMN15(4) BYTE PUBLIC AT (.BEGABSDATA + 25),
=           SYSMN16(4) BYTE PUBLIC AT (.BEGABSDATA + 29),
```

101

```
        =       SYSMN17(4) BYTE PUBLIC AT (.BEGABSDATA + 33),
        =       SYSMN18(4) BYTE PUBLIC AT (.BEGABSDATA + 37),
        =       SYSMN19(4) BYTE PUBLIC AT (.BEGABSDATA + 41),
        =       SYSMN23(4) BYTE PUBLIC AT (.BEGABSDATA + 45),
        =       SYSMN24(4) BYTE PUBLIC AT (.BEGABSDATA + 49),
        =       SYSMN25(4) BYTE PUBLIC AT (.BEGABSDATA + 53),
        =                   /* SYSTEM CONSOLE MESSAGES */
        =       EXTMSGLOCK BYTE PUBLIC AT (.BEGABSDATA + 57),
        =           /*
        =           Ø IF EXT. MSG BUFFER UNLOCKED
        =           OTHERWISE NUMBER OF CPTR CURRENTLY WORKING IN BUFFER
        =           */
        =       EXTMSG BYTE PUBLIC AT (.BEGABSDATA + 58),
        =           /*
        =           Ø IF NO EXTERNAL MSG TO PROCESS
        =           OTHERWISE NUMBER OF RECEIVING CPTR
        =           */
        =       EXTMSGIN BYTE PUBLIC AT (.BEGABSDATA + 59),
        =       EXTMSGOUT BYTE PUBLIC AT (.BEGABSDATA + 60),
        =           /*
        =           POINTER TO EXTERNAL MSG BUFFER */
        =       NUMEXTMSG BYTE PUBLIC AT (.BEGABSDATA + 61),
        =           /*
        =           NUMBER OF EXT. MSG IN EXT. MSG BUFFER
        =           */
        =       EXTLASTMSG BYTE PUBLIC AT (.BEGABSDATA + 62),
        =       EXTMSGBUFFER (512) BYTE PUBLIC AT (.BEGABSDATA + 63),
        =           /*
        =           EXTERNAL MESSAGE BUFFER
        =           */
        =       ENDABSDATA BYTE PUBLIC AT (.BEGABSDATA + 575);
        =           /*
        =           END OF ABSOLUTE DATA IN COMMON MEMORY
        =           */
  13  1 =   DCL LOADSBC BYTE PUBLIC AT (MSG$TBL$ADR - 4),
        =         START(3) BYTE PUBLIC AT (MSG$TBL$ADR - 3 );
```

```
                =           /*
                =           ABSOLUTE LOCATED VARIABLES USED FOR LOADING AND
                =           START OF SBC'S
                =           */
  14    1       =       DCL SAVESTACKPTR ADDRESS PUBLIC;
                =           /*
                =           STACK POINTER AT SYSTEM START
                =           */
  15    1       =       DCL CDCADR ADDRESS PUBLIC;
                =           /*
                =           ADDRESS TO BE CALLED AT CDC INTERRUPT
                =           */
  16    1       =       DCL EXCLEARBEG BYTE PUBLIC;
                =           /*
                =           BEGIN OF DATA TO BE CLEARED AT START
                =           */
  17    1       =       DCL (USART$IN,USART$OUT) BYTE PUBLIC;
  18    1       =       DCL CDCACTIVE BYTE PUBLIC;
                =           /* TRUE IF CDC INT ACTIVATED */
  19    1       =       DCL CODESAVE ADDRESS PUBLIC;
                =           /*
                =           CODE FOR MSG BUFFER OVERFLOW
                =           */
  20    1       =       DCL MSGBUFFER (MSGBUFLEN) BYTE PUBLIC,
                =           (MSGIN,MSGOUT) ADDRESS PUBLIC,
                =           NUMMSG BYTE PUBLIC,
                =           LASTMSG ADDRESS PUBLIC;
                =           /*
                =           MSGBUFFER IS A CIRCULAR FIFO LIST CONTAINING THE
                =             MSG WHICH ARE WAITING TO BE PROCESSED.
                =           MSGIN POINTS TO LOCATION OF NEXT MSG TO FILL IN.
                =           MSGOUT POINTS TO MSG TO BE PROCESSED NEXT.
                =           NUMMSG IS INCREMENTED WHEN A MSG IS SENT AND
                =             DECREMENTED WHEN A MSG IS PROCESSED.
                =             NUMMSG = 0: MSGBUFFER IS EMPTY.
                =           LASTMSG CONTAINS INDEX OF LAST BYTE OF LAST MSG
```

```
          =              IN MSGBUFFER + 1.
          =          */
 21   1   =    DCL PRIORLIST (MAXPRIOR) ADDRESS PUBLIC,
          =        PRIORSCHEDULE BYTE PUBLIC;
          =          /*
          =          PRIORLIST CONTAINS THE ADDRESSES OF THE ACTIVATED
          =          PRIORITY TASKS.
          =          A PRIORITY TASK IS CALLED WHEN IT IS SCHEDULED BY
          =          SETTING THE RESPECTIVE BIT IN PRIORSCHEDULE,
          =          E.G. IF BIT 3 (LSB = BIT Ø) IN PRIORSCHEDULE IS
          =          SET,THE ADDRESS IN PRIORLIST(3) IS CALLED.
          =          */
 22   1   =    DCL (XB1,XB2,XB3) BYTE PUBLIC,(XA1,XA2,XA3) ADDRESS PUBLIC;
          =          /*
          =          EXEC WORK VARIABLES
          =          */
 23   1   =    DCL PERXTBL (MAXPER) BYTE PUBLIC;
          =          /*
          =          PERXTBL CONTAINS POINTER TO PERLIST IN COMPACT FORM
          =          ØFFH - PERIODIC NOT ACTIVATED
          =          */
 24   1   =    DCL PERLIST (MAXPER) STRUCTURE (
          =             FREE BYTE,          /* TRUE - ITEM IS FREE */
          =             PERADR ADDRESS,     /* ADDRESS OF PERIODIC */
          =             PERINTADR ADDRESS, /* ADDRESS OF TIME INTERVAL */
          =             PERTIME (4) BYTE) PUBLIC, /* NEXT CALL TIME (RTC FORMAT) */
          =              /*
          =              PERLIST CONTAINS ALL SIGNIFICANT DATA OF AN
          =              ACTIVATED PERIODIC TASK
          =              */
          =          (PERINT BASED XA1) (5) BYTE,
          =              /*
          =              OVERLAY FOR TIME INTERVAL OF A PERIODIC
          =              */
          =          PERX BYTE PUBLIC,
          =              /*
```

```
            =           PERX IS SEARCH INDEX FOR PERLIST
            =           */
            =       ERRORMSG(17) BYTE PUBLIC,
            =           /*
            =           MESSAGE SEND BY ERROR ROUTINE.
            =           */
            =       NUMPER BYTE PUBLIC;
            =           /*
            =           NUMBER OF ACTIVATED PERIODICS
            =           */
 25   1     =   DCL EXCLEAREND BYTE PUBLIC;
            =           /*
            =           END OF DATA TO BE CLEARED AT START
            =           */
```

```
                $ EJECT
                $ INCLUDE(:F1:SCEXT.SRC)
          =     /*****************************************
          =     ******************************************

          =     : EXTERNAL DECLARATIONS OF SYSTEM ROUTINES
          =
          =     ******************************************/
          =
   26   1 =     FILL: PROCEDURE (P1,P2,P3) EXTERNAL;
   27   2 =           DCL (P1,P2) ADDRESS, P3 BYTE;
   28   2 =           END;
   29   1 =     CLEARDATA:PROCEDURE (P1,P2) EXTERNAL;
   30   2 =           DCL (P1,P2) ADDRESS;
   31   2 =           END;
   32   1 =     SHFT: PROCEDURE (P1) BYTE EXTERNAL;
   33   2 =           DCL P1 BYTE;
   34   2 =           END;
   35   1 =     BIT : PROCEDURE (P1,P2) BYTE EXTERNAL;
   36   2 =           DCL (P1,P2) BYTE;
   37   2 =           END;
   38   1 =     CLEARBIT: PROCEDURE (P1,P2) EXTERNAL;
   39   2 =           DCL P1 BYTE, P2 ADDRESS;
   40   2 =           END;
   41   1 =     SETBIT: PROCEDURE (P1,P2) EXTERNAL;
   42   2 =           DCL P1 BYTE, P2 ADDRESS;
   43   2 =           END;
   44   1 =     LEGAL:PROCEDURE (P1,P2,P3) BYTE EXTERNAL;
   45   2 =           DCL (P1,P2,P3) BYTE;
   46   2 =           END;
          =
   47   1 =     SETPERTIME:PROCEDURE (P1) EXTERNAL;
   48   2 =                 DCL P1 BYTE;
   49   2 =                 END;
   50   1 =     COPY$CLOCK:PROCEDURE (P1) EXTERNAL;
   51   2 =                 DCL P1 ADDRESS;
```

```
 52   2  =                 END;
 53   1  =    TIME$CMP : PROCEDURE (P1,P2) BYTE EXTERNAL;
 54   2  =                 DCL (P1,P2) ADDRESS;
 55   2  =                 END;
 56   1  =    PER$ACT :  PROCEDURE (P1,P2) BYTE EXTERNAL;
 57   2  =                 DCL (P1,P2) ADDRESS;
 58   2  =                 END;
 59   1  =    PER$CHG :  PROCEDURE (P1,P2) EXTERNAL;
 60   2  =                 DCL P1 BYTE, P2 ADDRESS;
 61   2  =                 END;
 62   1  =    PER$SUSP : PROCEDURE (P1) EXTERNAL;
 63   2  =                 DCL P1 BYTE;
 64   2  =                 END;
        =
 65   1  =    MSG$OVERFLOW: PROCEDURE EXTERNAL;
 66   2  =          END;
 67   1  =    PACK$MCB:PROCEDURE (P1) EXTERNAL;
 68   2  =          DCL P1 ADDRESS;
 69   2  =          END;
 70   1  =    SEND :PROCEDURE (P1) BYTE EXTERNAL;
 71   2  =          DCL P1 ADDRESS;
 72   2  =          END;
 73   1  =    GET  :PROCEDURE (P1) EXTERNAL;
 74   2  =          DCL P1 ADDRESS;
 75   2  =          END;
 76   1  =    RELEASE:PROCEDURE (P1) EXTERNAL;
 77   2  =          DCL P1 ADDRESS;
 78   2  =          END;
 79   1  =    SET$EXT$MSG:PROCEDURE EXTERNAL;
 80   2  =          END;
 81   1  =    REC$EXT:PROCEDURE EXTERNAL;
 82   2  =          END;
 83   1  =    SEND$EXT:PROCEDURE EXTERNAL;
 84   2  =          END;
 85   1  =    ILLEGAL$MSG:PROCEDURE EXTERNAL;
 86   2  =          END;
```

```
                =
 87   1     =    PRIORITY:PROCEDURE (P1) EXTERNAL;
 88   2     =             DCL P1 BYTE;
 89   2     =             END;
 90   1     =    ENTER$PRIOR: PROCEDURE (P1) BYTE EXTERNAL;
 91   2     =             DCL P1 ADDRESS;
 92   2     =             END;
 93   1     =    REM$PRIOR: PROCEDURE (P1) EXTERNAL;
 94   2     =             DCL P1 BYTE;
 95   2     =             END;
 96   1     =    SET$CDC:PROCEDURE (P1,P2) BYTE EXTERNAL;
 97   2     =             DCL (P1,P2) ADDRESS;
 98   2     =             END;
 99   1     =    ENTER$INT: PROCEDURE (P1) EXTERNAL;
100   2     =             DCL P1 BYTE;
101   2     =             END;
102   1     =    REM$INT:PROCEDURE (P1) EXTERNAL;
103   2     =             DCL P1 BYTE;
104   2     =             END;
                =
105   1     =    PRINT$ASYNC:PROCEDURE (P1,P2) EXTERNAL;
106   2     =                 DCL P1 ADDRESS, P2 BYTE;
107   2     =                 END;
108   1     =    PRINT$SYNC:PROCEDURE (P1,P2,P3) EXTERNAL;
109   2     =                 DCL P1 ADDRESS, (P2,P3) BYTE;
110   2     =                 END;
111   1     =    CON$INPUT$REQ:PROCEDURE (P1,P2) EXTERNAL;
112   2     =                 DCL (P1,P2) BYTE;
113   2     =                 END;
114   1     =    CON$LINK$REQ:PROCEDURE (P1) EXTERNAL;
115   2     =                 DCL P1 BYTE;
116   2     =                 END;
117   1     =    PAR$LINK$REQ:PROCEDURE (P1) EXTERNAL;
118   2     =                 DCL P1 BYTE;
119   2     =                 END;
120   1     =    CON$RELEASE: PROCEDURE EXTERNAL;
```

```
121   2  =                END;
122   1  =    CON$NEW$LINE:PROCEDURE (P1) EXTERNAL;
123   2  =                DCL P1 BYTE;
124   2  =          END;
125   1  =    CON$BEG$LINE:PROCEDURE (P1) EXTERNAL;
126   2  =                DCL P1 BYTE;
127   2  =                END;
128   1  =    PRINT$LINKED:PROCEDURE (P1,P2,P3,P4) EXTERNAL;
129   2  =                DCL P1 ADDRESS, (P2,P3,P4) BYTE;
130   2  =                END;
131   1  =    PAR$RELEASE: PROCEDURE EXTERNAL;
132   2  =                END;
133   1  =    WRITE: PROCEDURE (P1,P2,P3) EXTERNAL;
134   2  =                DCL P1 ADDRESS, (P2,P3) BYTE;
135   2  =                END;
136   1  =    WRITE$LINKED: PROCEDURE (P1,P2,P3,P4) EXTERNAL;
137   2  =                DCL P1 ADDRESS, (P2,P3,P4) BYTE;
138   2  =                END;
139   1  =    PAGE: PROCEDURE (P1) EXTERNAL;
140   2  =                DCL P1 BYTE;
141   2  =                END;
          =
142   1  =    ACTIVE: PROCEDURE (P1) BYTE EXTERNAL;
143   2  =            DCL P1 BYTE;
144   2  =            END;
145   1  =    UPD$STAT:PROCEDURE (P1,P2) EXTERNAL;
146   2  =            DCL (P1,P2) BYTE;
147   2  =            END;
148   1  =    CONV$ASC:PROCEDURE (P1) EXTERNAL;
149   2  =            DCL P1 BYTE;
150   2  =            END;
151   1  =    DEBLK : PROCEDURE EXTERNAL;
152   2  =            END;
153   1  =    GET$NUM:PROCEDURE BYTE EXTERNAL;
154   2  =            END;
155   1  =    VECTOR$ADD: PROCEDURE (P1,P2,P3) EXTERNAL;
```

```
156   2  =            DCL (P1,P2,P3) ADDRESS;
157   2  =            END;
158   1  =    VECTOR$SUB: PROCEDURE (P1,P2,P3) EXTERNAL;
159   2  =            DCL (P1,P2,P3) ADDRESS;
160   2  =            END;
161   1  =    NUMOUT: PROCEDURE (P1,P2,P3,P4,P5) EXTERNAL;
162   2  =            DCL (P1,P4) ADDRESS, (P2,P3,P5) BYTE;
163   2  =            END;
164   1  =    GET$CHAR:PROCEDURE BYTE EXTERNAL;
165   2  =            END;
           =
166   1  =    ENTER$MSG$MOD:PROCEDURE (P1,P2) EXTERNAL;
167   2  =             DCL P1 BYTE, P2 ADDRESS;
168   2  =             END;
169   1  =    MONITOR: PROCEDURE EXTERNAL;
170   2  =             END;
171   1  =    ERROR :  PROCEDURE (P1,P2,P3) EXTERNAL;
172   2  =             DCL P1 ADDRESS, (P2,P3) BYTE;
173   2  =             END;
```

```
                $EJECT

174   1         EXSTART:    PROCEDURE EXTERNAL;
175   2                   END EXSTART;

176   1         DCL SYSSTART LABEL PUBLIC; /* ENTRY POINT FOR RESTART PROCEDURE */

                /*
                ***************************************************************
                ***************************************************************
                **
                            EXECUTIVE
                **
                ***************************************************************
                */
177   1         EXEC:       PROCEDURE PUBLIC;
178   2                     DCL TMP$CLOCK (4) BYTE;/* TEMPORARY CLOCK VALUE */

179   2         EXEC0:      CALL EXSTART;
                          /* INITIALIZE */
                /*
                ***************************************************************
                **
                    PROCESS PRIORITY TASKS OF MODULES
                */
180   2         EXEC1:      DO WHILE PRIORSCHEDULE <> 0;
                          /* DO AS LONG AS PRIORITY CALLS ARE SCHEDULED */
181   3                   XB2 = 1;
182   3                   DO XB1 = 1 TO MAXPRIOR;
                            /* FIND HIGHEST PRIORITY SCHEDULED */
183   4                     XB3 = SCR(PRIORSCHEDULE,XB1);
184   4                     IF CARRY THEN
185   4                       DO;
186   5                       PRIORSCHEDULE = PRIORSCHEDULE XOR XB2;
                                /* RESET PRIORITY BIT */
187   5                       XA1 = PRIORLIST(XB1-1);
```

```
188   5                    CALL XA1;
                               /* CALL PRIORITY PROCEDURE */
189   5                    GO TO EXEC11;
190   5                    END;
191   4                  XB2 = ROL(XB2,1);
192   4                  END;
193   3     EXEC11:        ;
194   3                END;
            /*
            ***************************************************************
            **
                PROCESS PENDING REAL TIME MESSAGES
            */
195   2     EXEC2:         IF NUMMSG = 0 THEN GO TO EXEC21;
                         /* NO MSG TO PROCESS */
197   2                IF MSGOUT = LASTMSG THEN LASTMSG,MSGOUT = 0;
                         /* NEXT MSG AT TOP OF MSGBUFFER */
199   2                XA1 = MSGOUT;
200   2                MSGOUT = XA1 + MSGBUFFER(XA1+ML);
                         /* COMPUTE BEGIN OF NEXT MSG */
201   2                NUMMSG = NUMMSG - 1;
                         /* UPDATE NUMBER OF MSG */
202   2                ADRMSG = .MSGBUFFER(XA1);
203   2                ADRMSGDATA = ADRMSG + 4;
                          /* 'SET' MSG INTO WORKAREA */
204   2                IF MSG(RMN) <= LASTMN AND
                         MSG(RMN) >= FIRSTMN THEN DO;
206   3                  XA1 = MSG$MOD$ADDRESS(MSG(RMN));
207   3                  CALL XA1;
208   3                  END;
209   2                ELSE CALL SENDEXT;
210   2                GO TO EXEC1;
                         /* CHECK PRIORITY CALLS AGAIN */
211   2     EXEC21:    IF EXTMSG <> CPTR THEN GOTO EXEC3;
                         /* NO EXTERNAL MSG TO PROCESS */
213   2                CALL RECEXT;
```

```
                        /* TRANSFER MSG INTO OWN MSG BUFFER */
214   2                 GO TO EXEC1;
                        /* CHECK PRIORITY CALLS AGAIN */
                  /*
                  ***********************************************************
                  **
                      CHECK FOR PERIODIC CALLS OF MODULES
                  */
215   2           EXEC3:      ;
216   2                 IF NUMPER = 0 THEN GO TO EXEC4;
                        /* NO PERIODIC CALLS ACTIVE */
218   2                 XB1 = NUMPER - 1;
                        /* SET LIMIT FOR 1 SEARCH CYCLE */
219   2                 CALL COPY$CLOCK(.TMP$CLOCK);
                              /* COPY CLOCK VALUE */
220   2                 DO XB2 = 0 TO XB1;
                        /* CHECK ALL ACTIVATED PERIODICS */
221   3                   PERX = PERXTBL(XB2);
222   3                     IF TIME$CMP(.PERLIST(PERX).PERTIME,.TMP$CLOCK) THEN
                            /* CALL TIME REACHED */
223   3                     DO;
224   4                     XA1 = PERLIST(PERX).PERADR;
225   4                     CALL XA1;
                                /* CALL PERIODIC */
226   4                 CALL SETPERTIME(PERX); /* SET NEXT CALL TIME */
227   4                     GO TO EXEC1;
                              /* CHECK PRIORITY CALLS */
228   4                     END;
229   3                   END;
                  /*
                  ***********************************************************
                  **
                      BACKGROUND TASKS
                  */
230   2           EXEC4:      ;
231   2                 OUTPUT(0D6H) = TRUE; /* LIGHT LED FOR ONE MILISECOND */
```

```
232   2             GO TO EXEC1;
233   2             RETURN;
234   2             END EXEC;
```

```
                $ EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                        MAIN MODULE PROGRAM ENTRY
                **
                */
 235   1                  DISABLE;
 236   1        SYSSTART: SAVESTACKPTR = STACKPTR;
 237   1                  CPTR$ID = CPTR - 1 ; /* SET COMPUTER IDENTIFICATION */
                           /* SAVE STACKPOINTER FOR RESTART */
 238   1                  DO WHILE LOADSBC <> CPTR; /* WAIT FOR LOADING PROCESS */
 239   2                    END;

 240   1                DO WHILE START(CPTR-1) <> CPTR;
 241   2                   DO B1 = 0 TO 99; /* WAIT FOR 1000 MS */
 242   3                    CALL TIME(100);
 243   3                    END;
 244   2                   OUTPUT(0D6H) = TRUE; /* FLASH LED 1 MS. */
 245   2                   END;
 246   1                CALL EXEC;
 247   1             END;


MODULE INFORMATION:

     CODE AREA SIZE      = 01D4H     468D
     VARIABLE AREA SIZE  = 02A6H     678D
     MAXIMUM STACK SIZE  = 0004H       4D
     550 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE INTMSG
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:INTMSG.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            INTMSG:    DO;
                $ NOLIST
                $ INCLUDE(:F1:SEIODE.SRC)
                =   /*****************************
                =   *****************************************
                =   : EXTERNAL DECLARATIONS FOR PUBLIC
                =      SERIAL I/O INTERFACE PROCEDURES.
                =   *****************************************
                =   */
                =
 174    1   =   SEIO$START:      PROCEDURE EXTERNAL;
 175    2   =                     END;
 176    1   =   SE$PRINT$ASYNC:  PROCEDURE EXTERNAL;
 177    2   =                     END;
 178    1   =   SE$PRINT$SYNC :  PROCEDURE EXTERNAL;
 179    2   =                     END;
 180    1   =   SE$INPUT$REQ:    PROCEDURE EXTERNAL;
 181    2   =                     END;
 182    1   =   SE$INP$ACTIVATE:PROCEDURE EXTERNAL;
 183    2   =                     END;
 184    1   =   TERMIO:          PROCEDURE EXTERNAL;
 185    2   =                     END;
 186    1   =   SE$NEW$LINE:     PROCEDURE EXTERNAL;
 187    2   =                     END;
 188    1   =   SE$BEG$LINE:     PROCEDURE EXTERNAL;
 189    2   =                     END;
 190    1   =   SE$PRINT$LINKED:PROCEDURE EXTERNAL;
 191    2   =                     END;
```

```
                $EJECT
                $ INCLUDE(:F1:PAIODE.SRC)
            =   /*****************************************
            =   *****************************************
            =   : EXTERNAL DECLARATIONS OF PUBLIC PARALLEL
            =     OUTPUT INTERFACE PROCEDURES.
            =   *****************************************
            =   */
            =
192   1     =   DCL PRELL$OUT BYTE EXTERNAL;
            =
193   1     =   PAIO$START:      PROCEDURE EXTERNAL;
194   2     =                    END;
195   1     =   PA$OUT$ACTIVATE:PROCEDURE EXTERNAL;
196   2     =                    END;
197   1     =   PA$OUT$RELEASE: PROCEDURE EXTERNAL;
198   2     =                    END;
199   1     =   PA$PRINT:        PROCEDURE EXTERNAL;
200   2     =                    END;
201   1     =   PA$NEW$PAGE:     PROCEDURE EXTERNAL;
202   2     =                    END;
203   1     =   PA$PRINT$LINKED:PROCEDURE EXTERNAL;
204   2     =                    END;
                $ INCLUDE(:F1:MONIDE.SRC)
            =   /*************************************
            =   *************************************
            =   : EXTERNAL DECLARATIONS OF MONITOR
            =     PUBLIC PROCEDURES.
            =   *************************************
            =   */
            =
205   1     =   MONITOR$TRAP:PROCEDURE EXTERNAL;
206   2     =                END;
207   1     =   MONI$INTER: PROCEDURE EXTERNAL;
208   2     =                END;
209   1     =   MONI$INIT:  PROCEDURE EXTERNAL;
```

```
210   2  =                    END;
211   1  =    MONI$ENDER:  PROCEDURE EXTERNAL;
212   2  =                    END;
213   1  =    PRINT$LINE:  PROCEDURE EXTERNAL;
214   2  =                    END;
```

```
                $EJECT
                /*
                ****************************************************************
                **
                **                  INTERRUPT PROCESSING
                */
                /*
                ****************************************************************
                */
215   1         INTRESET:   PROCEDURE;

216   2                     OUTPUT(0DAH) = 00100000B;
217   2                     RETURN;
218   2                     END INTRESET;
                /*
                ****************************************************************
                */
219   1         INT0:       PROCEDURE INTERRUPT 0;

                              /* CDC INTERRUPT */

220   2                     IF NOT CDCACTIVE THEN RETURN;
222   2                     OUTPUT(0DFH) = 01110000B;/* CNT 1 MODE 0 */
223   2                     CALL CDCADR;
                                /* CALL PROCESS OF INTERRUPT */
224   2                     INTMASK = INTMASK OR 1;
225   2                     OUTPUT(0DBH) = INTMASK;
                                /* DISABLE INT0 */
226   2                     CALL INTRESET;
227   2                     CDCACTIVE = FALSE;
228   2                     RETURN;
229   2                     END INT0;
                /*
                ****************************************************************
                */
230   1         INT1:             PROCEDURE INTERRUPT 1;
```

```
231   2                     CALL MONITOR$TRAP;

232   2                     CALL INTRESET;
233   2                     RETURN;
234   2                     END INT1;
              /*
              ***************************************************************
              */
235   1       INT2:         PROCEDURE INTERRUPT 2;
236   2                 DCL INCRE BYTE DATA(1);

                        /* RTC INTERRUPT */

237   2                   IF CLOCK = 0 THEN DO;
239   3                       CLOCK = 0FFH;
240   3                       RTC3 = RTC3 + INCRE    ;
241   3                       RTC2 = RTC2 PLUS 0 ;
242   3                       RTC1 = RTC1 PLUS 0 ;
243   3                       RTC0 = RTC0 PLUS 0 ;
244   3                       CLOCK = 0;
                          /* UPDATE RTC */
245   3                     END;

246   2                 OUTPUT(0DCH) = 33H;
247   2                 OUTPUT(0DCH) = 4;
                            /* 930 NS X 433H (1075) = .99975 MS */
                            /* SET CTR 0 TO 1 MSEC IN MODE 0 */

248   2                 CALL INTRESET;
249   2                 RETURN;
250   2                 END INT2;
              /*
              ***************************************************************
              */
251   1       INT3:         PROCEDURE INTERRUPT 3;
```

```
252   2                 CALL PRIORITY(USART$IN);
                            /* SCHEDULE PRIORITY CALL */
253   2                 CALL INTRESET;
254   2                 RETURN;
255   2                 END INT3;
                  /*
                  ****************************************************************
                  */
256   1         INT4:       PROCEDURE INTERRUPT 4;

257   2                 CALL PRIORITY(USART$OUT);
                            /* SCHEDULE PRIORITY CALL */
258   2                 CALL INTRESET;
259   2                 RETURN;
260   2                 END INT4;
                  /*
                  ****************************************************************
                  */
261   1         INT5:       PROCEDURE INTERRUPT 5;

262   2                       CALL PRIORITY(PRELL$OUT);
263   2                 CALL INTRESET;
264   2                 RETURN;
265   2                 END INT5;

                  /*
                  ****************************************************************
                  */

266   1         DCL SYSSTART LABEL EXTERNAL; /* RESTART ENTRY POINT */

267   1         EXEC:       PROCEDURE EXTERNAL;
268   2                 END EXEC;

269   1         INT6:       PROCEDURE; /* RESTART INTERRUPT */
```

```
270   2             STACKPTR = SAVESTACKPTR;
                        /* RESET STACK POINTER */
271   2             LOADSBC = 1; /* COMPUTER 1 BEGINS PROCESS */
272   2             GO TO SYSSTART;

273   2                     END INT6;
```

```
$ EJECT
/*
*****************************************************************
*****************************************************************

                    SYSTEM MESSAGE TASK

RECEIVES REQUEST MESSAGES FOR THE USE OF THE I/O FACILITIES
 AND THE MONITOR.
*****************************************************************
*/
274   1     MSGENT0: PROCEDURE PUBLIC;
275   2        IF MSG(MN) < 10  OR  MSG(MN) > 26    /* CHECK BOUNDS */
                 THEN CALL ILLEGAL$MSG;
277   2           ELSE DO CASE MSG(MN)-10 ; /* SWITCH TO CORRESPONDING ACTION */

278   3                   CALL SE$PRINT$ASYNC ; /* MN 10 */
279   3                   CALL SE$PRINT$SYNC  ; /* MN 11 */
280   3                   CALL SE$INPUT$REQ   ; /* MN 12 */
281   3                   CALL SE$INP$ACTIVATE; /* MN 13 */
282   3                   CALL PA$OUT$ACTIVATE; /* MN 14 */
283   3                   CALL TERMIO         ; /* MN 15 */
284   3                   CALL SE$NEW$LINE    ; /* MN 16 */
285   3                   CALL SE$BEG$LINE    ; /* MN 17 */
286   3                   CALL SE$PRINT$LINKED; /* MN 18 */
287   3                   CALL PA$OUT$RELEASE ; /* MN 19 */
288   3                   CALL MONI$INTER     ; /* MN 20 */
289   3                   CALL MONI$INIT      ; /* MN 21 */
290   3                   CALL MONI$ENDER     ; /* MN 22 */
291   3                   CALL PA$PRINT       ; /* MN 23 */
292   3                   CALL PA$NEW$PAGE    ; /* MN 24 */
293   3                   CALL PA$PRINT$LINKED; /* MN 25 */
294   3                   CALL PRINT$LINE     ; /* MN 26 */
295   3                   END;
296   2           RETURN;
297   2           END MSGENT0;
```

```
                $ EJECT
                /*
                ***************************************************************
                 SETS EXECUTIVE DATA STRUCTURES.
                */
298   1         SET$EX$DATA: PROCEDURE;
299   2             DCL  ERM(7) BYTE DATA('ERROR: ');

300   2                       CALL CLEAR$DATA(.EX$CLEAR$BEG,.EX$CLEAR$END);
301   2                       CALL MOVE(7,.ERM,.ERRORMSG);
302   2                       CALL ENTER$MSG$MOD(EX,.MSGENT0);
303   2                       CALL UPDSTAT(EX,0BH); /* ACTIVATE EXEC. MSG. HANDLER */
304   2                       DO B1 = 0 TO LAST(PERLIST); /* INIT PERIODIC LIST */
305   3                          PERLIST(B1).FREE = TRUE ;
306   3                          END;
307   2                       ADRSTAT = .MODSTATUS(0) + FIRSTMN ; /* SET OVERLAY */
308   2                       IF CPTR = 1 THEN CALL CLEAR$DATA(.BEGABSDATA,.ENDABSDATA);
310   2                       RETURN;
311   2                       END;
```

```
                $ EJECT
                /*
                ***************************************************************
                 SETS INITIAL SYSTEM MESSAGE STATES
                */
  312   1       SET$EX$MSGS: PROCEDURE;
  313   2                    DCL SYSMN0(4) BYTE ; /* START MESSAGE */
  314   2                    NUMMSG = 0 ; /* INIT NUMBER OF MESSAGES FLAG */
  315   2                    SYSMN10(2)=10;SYSMN11(2)=11;SYSMN12(2)=12;
  318   2                    SYSMN13(2)=13;SYSMN14(2)=14;SYSMN15(2)=15;
  321   2                    SYSMN16(2)=16;SYSMN17(2)=17;SYSMN18(2)=18;
  324   2                    SYSMN19(2)=19;SYSMN23(2)=23;SYSMN24(2)=24;
  327   2                    SYSMN25(2)=25;
  328   2                    SYSMN12(3)=5;
  329   2                    SYSMN13(3),SYSMN14(3),SYSMN15(3),
                              SYSMN16(3),SYSMN17(3),SYSMN19(3),SYSMN24(3)=4;
  330   2                    SYSMN0(1)=EX;SYSMN0(2)=0;SYSMN0(3)=4;
  333   2                    DO B1 = 1 TO LAST(MODSTAT);/* SEND INIT MSG IF PRESENT */
  334   3                       IF MODSTAT(B1) <> 0 THEN DO;
  336   4                                                SYSMN0(0) = B1 + FIRSTMN;
  337   4                                                CALL PACKMCB(.SYSMN0);
  338   4                                                NUMMSG = NUMMSG + 1 ;
  339   4                                                END;
  340   3                       END;
  341   2                    RETURN;
  342   2                    END;
```

```
                $ EJECT
                /*
                ***************************************************************
                 SETS INITIAL INTERRUPT STATE
                */
343   1         SET$EX$INTE: PROCEDURE;
                           /* INITIALIZE INTERRUPT VECTOR */
344   2                    INT0$LOC,INT1$LOC,INT2$LOC,INT3$LOC,
                           INT4$LOC,INT5$LOC,INT6$LOC,INT7$LOC = 0C3H;/* JMP */
345   2                    INT0$PRO = .INT0 ; INT1$PRO = .INT1 ;
347   2                    INT2$PRO = .INT2 ; INT3$PRO = .INT3 ;
349   2                    INT4$PRO = .INT4 ; INT5$PRO = .INT5 ;
351   2                    INT6$PRO = .INT6 ; INT7$PRO = .INT6 ;
353   2                    A4 = INTVECTOR ;
354   2                    B1 = BL AND 11100000B;
355   2                    OUTPUT(0DAH) = B1 OR 00010010B; /* ICW1 */
356   2                    OUTPUT(0DBH) = BU ;             /* ICW2 */
357   2                    INTMASK = 10111111B ; /* ONLY RESET INTERRUPT SET */
358   2                    RETURN;
359   2                    END;
```

```
$ EJECT
/*
**********************************************************************
**********************************************************************
**
                                                              START

    INITIALIZE COMPUTER
**
**********************************************************************
*/

360   1   EXSTART:    PROCEDURE PUBLIC;
361   2           DCL SYS$TAG(*) BYTE DATA('RDY COMPUTER ',CPTR+30H,0AH,0DH);

362   2           DISABLE;
363   2               CALL SET$EX$DATA;
364   2               CALL SET$EX$MSGS;
365   2               CALL SET$EX$INTE;
366   2               OUTPUT(0DFH) = 00110000B; /* CNT 0 MODE 0 */
367   2               OUTPUT(0DFH) = 01110000B; /* CNT 1 MODE 0 */
368   2           IF CPTR = 1 THEN
369   2             DO;
370   3                 INTMASK = INTMASK XOR 00000100B;/* RTC INTERR. */
371   3                 OUTPUT(0DCH) = 33H;
372   3                 OUTPUT(0DCH) = 4; /* LOAD COUNTER 0 WITH 1 MSEC 433H */
373   3             END;
374   2               LOADSBC = LOADSBC + 1 ; /* TRIGGER NEXT COMPUTER */

375   2           CALL SEIO$START; /* INITIALIZE SERIAL IO MODULE */
376   2           CALL PAIO$START; /* INITIALIZE PARALLEL OUTPUT MODULE */
377   2           OUTPUT(0DBH) = INTMASK; /* INIT INTERRUPT MASK */
378   2           CALL PRINT$ASYNC(.SYS$TAG,LENGTH(SYS$TAG));
379   2           ENABLE;

380   2           RETURN;
```

```
 381   2              END EXSTART;
 382   1      END INTMSG;
```

MODULE INFORMATION:

```
     CODE AREA SIZE     = 0382H    898D
     VARIABLE AREA SIZE = 0004H      4D
     MAXIMUM STACK SIZE = 000AH     10D
     627 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SEIOMOD
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SEIO.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            SE$IO$MOD: DO;
                $NOLIST
                /*
                ***************************************************************
                **
                                        SERIAL IO DATA
                **
                ***************************************************************
                **
                **                  CRT I/O PORTS AND USART CONTROL WORDS
                */
  26    1       DCL CRTDATAIN LIT '0FCH',       /* DATA INPUT */
                    CRTSTATIN LIT '0EDH',       /* STATUS INPUT */
                    CRTDATAOUT LIT '0ECH',      /* DATA OUTPUT */
                    CRTCMDOUT LIT '0EDH',       /* COMMAND OUTPUT */
                    TMCP    LIT '0DFH',      /* TIMER COMM. PORT */
                    CTR2    LIT '0DEH',     /* COUNTER 2 PORT */
                    B2400   LIT '01CH',
                    MCW LIT '01001110B',        /* MODE CONTROL WORD */
                    CCW1 LIT '00100111B',        /* COMMAND CONTROL WORD */
                    CCW2 LIT '00100110B',       /* COMMAND CONTROL WORD 2 */
                    RESET LIT '01000000B';      /* RESET CRT */
                /*
                ***************************************************************
                **
                **                          CRT CONTROL CHARACTERS
                */
  27    1       DCL HOME LIT '2',               /* HOME CURSOR */
                    FC LIT '1CH',               /* FORWARD CURSOR */
                    BC LIT '8',                 /* BACK CURSOR */
```

```
                UC LIT '1AH',               /* UP CURSOR */
                CR LIT '0DH',               /* CARRIAGE RETURN */
                LF LIT '0AH',               /* LINE FEED */
                CLEAR LIT '1EH',            /* CLEAR SCREEN */
                ADRXY LIT '0CH',            /* START X/Y ADDRESSING */
                POLL LIT '1DH',             /* ROLL MODE */
                BELL LIT '7',
                CONSOL$ID LIT '15H',         /* CONSOL IDENTIFICATION SIGNAL (NAK) */
                MONITOR$ID LIT '14H',        /* MONITOR REQUEST ID SIGNAL (DC4) */
                CLRLINE LIT '17H',          /* CLEAR LINE */
                BOL LIT '96',               /* BEGIN OF LINE */
                ASYNCLINE LIT '102',        /* LINE FOR ASYNCHRONOUS PRINTS */
                INPLINE LIT '119',          /* INPUT LINE */
                ARROW LIT '5EH',            /* UP ARROW */
                BLANK LIT ''' ''',          /* BLANK */
                LINELENGTH LIT '79',        /* LINE LENGTH */
                CRTOUTLEN LIT '256',        /* LENGTH OF OUTPUT BUFFER */
                DELINPUT LIT '7FH';         /* DELETE LAST INPUT (RUB OUT) */
130             /*
                ***************************************************************
                **
                **                  MESSAGE CONTROL BLOCKS
                */
28    1         DCL CSMN20(4) BYTE  ,
                      /* TRANSFER INPUT TEXT MSG */
                    CSMN21(4) BYTE  ,
                      /* INPUT REQUEST TELL BACK MSG
                           1ST DATA BYTE: TRUE - REQUEST ACKNOWLEDGED
                                          FALSE OTHERWISE */
                    CSMN22(4) BYTE  ,
                      /* TERMINATE I/O MSG
                         SEND TO MODULE CONNECTED TO CRT IF INPUT IS 'INT' */
                    CSMN26(4) BYTE  ;
                      /* TELLBACK MSG */
                /*
                ***************************************************************
```

```
                **
                **                      CONSTANT DATA
                **
                */
  29    1       DCL ROLLSCR (8) BYTE DATA (CR,ADRXY,BOL,ASYNCLINE+1,CLRLINE,ADRXY,BOL,
                                           INPLINE),
                        /* ROLL SCREEN,CLEAR LINE AFTER ASYNCLINE,RETURN TO
                                BEGIN OF INPUT LINE */
                    NEWL(2) BYTE DATA (1,' '),
                    BEGL(4) BYTE DATA (0,ADRXY,BOL,INPLINE),
                    ASYNC1(10) BYTE DATA (ARROW,ADRXY,BOL,ASYNCLINE,CLRLINE,
                                          BELL,'*** ');
                        /* START ASYNCHRONOUS PRINT */
                /*
                ***************************************************************
                **
                **                      VARIABLE DATA
                **
                */
  30    1       DCL CSBEGDATA BYTE,
                    INACTMOD BYTE,
                        /* MODULE WITH CURRENT INPUT ACTIVE */
                    CSTBCODE BYTE, /* REQUESTED TELLBACK CODE */
                    CRTIN(LINELENGTH) BYTE,
                        /* CRT INPUT BUFFER */
                    CRTOUT(CRTOUTLEN) BYTE,
                        /* CRT OUTPUT BUFFER */
                    (INX,OUTX) BYTE,
                        /* POINTER TO CRTIN AND CRTOUT (NEXT TO FILL) */
                    (CURSORINP,CURSORSAVE) BYTE,
                        /* CURSOR POSITION ON INPUT LINE */
                    OUTACTIVE BYTE,
                            /* TRUE IF OUTPUT ACTIVE */
                    INPREQUEST BYTE,
                        /* TRUE IF INPUT REQUEST MSG RECEIVED */
                    OUTPTR BYTE,
                        /* POINTER TO NEXT CHARACTER TO PRINT */
                    ROLLSCREEN BYTE,
```

131

```
    /* TRUE IF SCREEN IS TO ROLL AFTER INPUT */
N$OF$RATES BYTE,
    /* NUMBER OF RATES TESTED */
BAUD$RATE ADDRESS,
    /* BAUD RATE BEEN TESTED */
CHAR BYTE,
    /* INPUT CHARACTER */
CSTELLBACK BYTE,
    /* TRUE IF TELLBACK MSG REQUIRED AFTER SYNC OUTPUT */
CSENDDATA BYTE;
    /* END OF VARIABLE DATA */
```

132

```
                $EJECT
                $NOLIST
                /*
                ****************************************************************************
                ****************************************************************************
                **
                                                                        CSCONVXY

                    CONVERT INPUT TO XY-ADDRESSING
                **
                ****************************************************************************
                */
179     1       CSCONVXY:    PROCEDURE (B) BYTE;

180     2       DCL     (A,B) BYTE;

181     2               A =  B + 95;
182     2               IF B < 33 THEN RETURN A;
184     2               A = A - 64;
185     2               IF B < 65 THEN RETURN A;
187     2               A = A - 64;
188     2               RETURN A;
189     2               END CSCONVXY;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                    PACKCRTOUT
                    PACK INPUT INTO CRT OUTPUT BUFFER
                **
                ***************************************************************
                */
190    1        PACKCRTOUT: PROCEDURE(B);

191    2        DCL     B BYTE;

192    2                IF OUTX > CRTOUTLEN THEN RETURN;
                            /* OVERFLOW */
194    2                CRTOUT(OUTX) = B;
195    2                OUTX = OUTX + 1;
196    2                RETURN;
197    2                END PACKCRTOUT;
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                        PACK
                       PACK CODE FOR SCREEN
                **
                ***************************************************************
                */
198    1        PACK: PROCEDURE(A,B) ;
199    2           DCL A ADDRESS,B BYTE,(C BASED A) (1) BYTE;

200    2           DO B1 = 0 TO B ;
201    3              CALL PACKCRTOUT(C(B1));
202    3           END;
203    2           RETURN;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                    CSOUTPUT
                    CONTINUE CRT OUTPUT
                    PRIORITY PROCEDURE , CALL SCHEDULED BY INT 3
                    PROCESS LEVEL 3 CRT OUTPUT INTERRUPT
                x***************************************************************
                */
205   1         CSOUTPUT:   PROCEDURE;

206   2                 IF OUTPTR >= OUTX THEN
                            /* NO MORE CHARACTERS FOR OUTPUT */
207   2                   DO;
208   3                   OUTPTR = 0;OUTX = 0;
210   3                   OUTACTIVE = FALSE;
211   3                   IF CSTELLBACK THEN
                              /* TELLBACK MSG REQUIRED */
212   3                     DO;
213   4                     CSTELLBACK = FALSE;
214   4                     CSMN26(0) = INACTMOD;
215   4                   IF NOT SEND(.CSMN26)
                           THEN CALL ERROR(.CSOUTPUT,0,15);
217   4                    ELSE MSGDATA(0) = CSTBCODE;
                                /* SEND REQUIRED MSG AND TELLBACK CODE */
218   4                     END;
219   3                   DO WHILE (INPUT(CRTSTATIN) AND 4) <> 4 ; END;
221   3                      OUTPUT(CRTCMDOUT) = CCW2 ; /* DISABLE T*E */
222   3                   RETURN;
223   3                   END;
224   2                 OUTPUT(CRTDATAOUT) = CRTOUT(OUTPTR);
                            /* OUTPUT NEXT CHARACTER */
225   2                 OUTPTR = OUTPTR + 1;
226   2                 RETURN;
```

```
                $EJECT
                /*****************************************************************
                ******************************************************************
                **
                                                                    STARTOUT
                    START CRT OUTPUT IF NO OUTPUT ACTIVE
                ******************************************************************
                */
228    1        STARTOUT: PROCEDURE;

229    2                 IF OUTACTIVE THEN RETURN;
231    2                     OUTPUT(CRTCMDOUT) = CCW1; /* ENABLE T*E */
232    2                 OUTACTIVE = TRUE;
233    2                 RETURN;
234    2                 END STARTOUT;
                /*
                ******************************************************************
                ******************************************************************
                **
                                                                      TERMIO
                    TERMINATE I/O
                ******************************************************************
                */
235    1        TERMIO: PROCEDURE PUBLIC;

236    2                 IF INACTMOD = 0FFH THEN RETURN;
                              /* NO MODULE WITH ACTIVE I/O */
238    2                 CSMN22(0) = INACTMOD;
239    2                 IF NOT SEND(.CSMN22(0)) THEN RETURN;
                              /* SEND TERMINATE MSG */
241    2                 INACTMOD = 0FFH;
242    2                 INPREQUEST = FALSE;
243    2                     CALL PACK(.ROLLSCR,LAST(ROLLSCR));
244    2                     CURSORINP,CURSORSAVE = 1 ;
245    2                 CALL STARTOUT;
246    2                 RETURN;
```

```
                $EJECT
                /*
                ****************************************************************
                ****************************************************************
                **
                                                                    CSINPUT

                    CONTINUE CRT INPUT
                    PRIORITY PROCEDURE , CALL SCHEDULED BY INT 3
                    PROCESS LEVEL 3 CRT INPUT INTERRUPT
                **
                ****************************************************************
                */
248   1         CSINPUT:    PROCEDURE;
249   2             DCL EOT LIT '04H'; /* END OF TEXT ID */

250   2                 CHAR = INPUT(CRTDATAIN) AND 07FH;
                          /* GET NEXT CHARACTER */

251   2                 IF CHAR = CONSOL$ID THEN DO;
                            /* CONSOL IDENTIFICATION */
253   3                   SYSMN10(0),SYSMN11(0),SYSMN12(0),
                          SYSMN13(0),SYSMN15(0),SYSMN16(0),
                          SYSMN17(0),SYSMN18(0) = EX;
                                  /* IDENTIFY CONSOLE */
254   3                   RETURN;
255   3                   END;

256   2                 IF CHAR = MONITOR$ID THEN
                          /* CRT INTERRUPTION */
257   2                 DO;
258   3                 CALL TERMIO;
259   3                     CALL MONITOR ; /* ENTER MONITOR */
260   3                     RETURN;
261   3                 END;
```

```
262   2          IF OUTACTIVE OR NOT INPREQUEST THEN RETURN;

264   2              IF CHAR = DELINPUT THEN
                         /* DELETE LAST CHARACTER */
265   2                DO;
266   3                IF CURSORINP <= CURSORSAVE
                         THEN CALL PACKCRTOUT(BELL);
268   3                ELSE DO;
269   4                   CALL PACKCRTOUT(BC);
270   4                   CALL PACKCRTOUT(CLRLINE);
271   4                   INX = INX - 1;
272   4                   CURSORINP = CURSORINP - 1;
273   4                   END;
274   3                   CALL STARTOUT;
275   3                   RETURN;
276   3                END;

277   2              IF CHAR = CR THEN
                         /* END OF INPUT */
278   2                DO;
279   3                CSMN20(0) = INACTMOD;
                             /* SET RECEIVING MODULE */
280   3                CSMN20(3) = INX + 4;
                             /* MSG LENGTH */
281   3                INPREQUEST = FALSE;
282   3                IF SEND(.CSMN20(0)) THEN
283   3                      DO;CRTIN(INX) = EOT;
285   4                         CALL MOVE(INX,.CRTIN(1),ADRMSGDATA);
286   4                         END;
287   3                  ELSE INACTMOD = 0FFH;
                             /* RECEIVING MODULE NO LONGER ACTIVE */
288   3                INX = 1;
289   3                IF ROLLSCREEN THEN
290   3                  DO;
291   4                     CALL PACK(.ROLLSCR,LAST(ROLLSCR));
292   4                     CURSORINP,CURSORSAVE=1 ;
```

```
 293   4                      CALL STARTOUT;
 294   4                   END;
 295   3                     RETURN;
 296   3                END;


 297   2           IF CHAR < 20H OR CHAR > 5AH OR CURSORINP>=LAST(CRTIN)
                      THEN DO; /* INVALID CHARACTER */
 299   3                     CALL PACKCRTOUT('?');
 300   3                     CALL PACKCRTOUT(BELL);
 301   3                     CALL PACKCRTOUT(BC);
 302   3                     END;
 303   2                 ELSE DO;
 304   3                     CALL PACKCRTOUT(CHAR);
 305   3                     CRTIN(INX) = CHAR;
 306   3                     INX = INX + 1 ;
 307   3                     CURSORINP = CURSORINP + 1 ;
 308   3                     END;
 309   2                        CALL STARTOUT;
 310   2                        RETURN;
 311   2                 END CSINPUT;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                SEPRINTASYNC

                    PRINT ASYNCHRONOUS TEXT     (MN 10)
                    ( MAX 1 LINE )
                **
                ***************************************************************
                */
312    1        SEPRINTASYNC: PROCEDURE PUBLIC ;
313    2           DCL ASYNC2(3) BYTE;

314    2             ASYNC2(0) = ADRXY ; ASYNC2(2) = INPLINE ;
316    2                    CALL PACK(.ASYNC1,LAST(ASYNC1));
317    2                      CALL PACK(ADRMSGDATA,MSG(ML)-5);
                            /* PACK MSG INTO OUTPUT BUFFER */
318    2                 ASYNC2(1) = CSCONVXY(CURSORINP);
                            /* RETURN CURSOR TO LAST INPUT POSITION */
319    2                    CALL PACK(.ASYNC2,LAST(ASYNC2));
320    2                 CALL STARTOUT;
321    2                 RETURN;
322    2                 END SEPRINTASYNC;
```

```
                $EJECT
                /*
                **********************************************************
                **********************************************************
                **
                                                              NOT$ACT
                        CHECK IF MSG SENDER IS ACTIVE
                **
                **********************************************************
                */
323    1        NOT$ACT: PROCEDURE BYTE ;
324    2           IF MSG(SMN) <> INACTMOD THEN DO;
326    3              CALL ILLEGALMSG;
327    3              RETURN TRUE;
328    3              END;
329    2           RETURN FALSE;
330    2           END NOT$ACT;
                /*
                **********************************************************
                **********************************************************
                **
                                                           SE$INPUT$REQ
                        REQUEST INPUTT FROM DEVICE
                        IF MSGDATA(0) = TRUE -> ROLL SCREEN
                *
                **********************************************************
                */
331    1        SE$INPUT$REQ: PROCEDURE PUBLIC;
332    2                 IF NOT$ACT THEN RETURN;
334    2                 INPREQUEST = TRUE ;
335    2                 ROLLSCREEN = MSGDATA(0) ;
336    2                 CURSORSAVE = CURSORINP ;
337    2                 RETURN;
338    2                 END;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                SEPRINTSYNC

                    PRINT SYNCHRONOUS TEXT     (MN 11 AND MN 18)
                    PRINT ON INPUT LINE
                    IF MSGDATA(0) = TRUE : ROLL SCREEN AFTER OUTPUT
                **
                ***************************************************************
                */
339   1         SEPRINTSYNC: PROCEDURE PUBLIC;

340   2                       IF NOT$ACT THEN RETURN;
                              /* MSG NOT FROM CORRECT MODULE */
342   2                       CALL PACK(ADRMSGDATA+1,MSG(ML)-6);
                              /* MOVE CHARACTERS FROM MSG TO CRT OUTPUT BUFFER */
343   2                   IF MSGDATA(0)
                              THEN CALL PACK(.ROLLSCR,LAST(ROLLSCR));
                              /* ROLL SCREEN AFTER OUTPUT */
345   2                     ELSE CURSORINP = CURSORINP + MSG(ML) - 5;
346   2                   CALL STARTOUT;
347   2                   RETURN;
348   2                   END SEPRINTSYNC;
```

```
                $EJECT
                /*
                :**********************************************************
                ********** **********************************************
                **
                                                              SEPRINTLINKED
                PRINT MSG AND SEND BACK A CODE
                **
                **********************************************************
                */
349   1         SE$PRINT$LINKED: PROCEDURE PUBLIC;

350   2                            CSTELLBACK = TRUE;
351   2                            CSTBCODE = MSGDATA(0); /* SET TELLBACK CODE */
352   2                            MSG(ML) = MSG(ML) - 1;/* READJUST MSG LENGHT */
353   2                            ADRMSGDATA=ADRMSGDATA+1;
354   2                            CALL SE$PRINT$SYNC; /* PRINT MSG */
355   2                            RETURN;
356   2                            END;
```

```
                $EJECT
                /*
                ****************************************************************
                ****************************************************************
                **
                                                                SEINPACTIVATE

                    ACTIVATE INPUT FOR SENDING MODULE     (MN 13)
                **
                ****************************************************************
                */
357   1         SEINPACTIVATE: PROCEDURE PUBLIC;

358   2                 CSMN21(0) = MSG(SMN);
359   2                 IF NOT SEND(.CSMN21(0)) THEN RETURN;
                            /* SEND ACKNOWLEDGE MSG BACK */
361   2                 IF INACTMOD <> 0FFH
                        THEN MSGDATA(0) = FALSE;
                             /* CRT NOT FREE */
363   2                   ELSE DO;
364   3                     MSGDATA(0) = TRUE;
365   3                     INACTMOD = MSG(SMN);
366   3                     INPREQUEST = FALSE;
367   3                     END;
368   2                 RETURN;
369   2                 END SEINPACTIVATE;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                POS$CUR

                        POSITION CURSOR
                **
                ***************************************************************
                */
370    1        POS$CUR: PROCEDURE (A,B);
371    2            DCL A ADDRESS,B BYTE;

372    2            MSG(ML) = B ;
373    2            ADRMSGDATA = A ;
374    2            CALL SEPRINTSYNC ;
375    2            RETURN;
376    2            END POS$CUR;
                /*
                ***************************************************************
                ***************************************************************
                **
                                                          SE$NEW$LINE
                *
                ***************************************************************
                */
377    1        SE$NEW$LINE: PROCEDURE PUBLIC;
378    2                     CALL POS$CUR(.NEWL,6);
379    2                     RETURN;
380    2                     END;
```

```
                $EJECT
                /*
                **********************************************************
                **********************************************************
                **
                                                        SE$BEG$LINE
                **
                **********************************************************
                */
381    1        SE$BEG$LINE: PROCEDURE PUBLIC;
382    2                     CALL POS$CUR(.BEGL,8);
383    2                     CURSORINP,CURSORSAVE = 1 ;
384    2                     RETURN;
385    2                     END;
                /*
                **********************************************************
                **********************************************************
                **
                INIT$USART: INITIALIZE USART BAUD RATE
                *
                **********************************************************
                */
386    1        INIT$USART: PROCEDURE;
387    2            OUTPUT(CRTCMDOUT) = RESET;
388    2            OUTPUT(CRTCMDOUT) = MCW ;
389    2            OUTPUT(CRTCMDOUT) = 36H;
390    2            BAUD$RATE = B2400;
391    2            OUTPUT(TMCP) = 10110110B; /* COUNTER 2 MODE 3 */
392    2            OUTPUT(CTR2) = LOW(BAUD$RATE);
393    2            OUTPUT(CTR2) = HIGH(BAUD$RATE);
394    2            RETURN;
395    2            END;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                    SEIOSTART

                    INITIALIZE CRT MODULE    (MN 00)
                **
                ***************************************************************
                */
396   1         SEIOSTART:   PROCEDURE PUBLIC;

397   2                        CSMN20(1),CSMN21(1),CSMN22(1),CSMN26(1) = EX;
398   2                        CSMN20(2)=20;CSMN21(2)=21;CSMN22(2)=22;CSMN26(2)=26;
402   2                        CSMN21(3),CSMN26(3) = 5 ; CSMN22(3) = 4 ;
404   2                   CALL INIT$USART;
                              /* MODE CONTROL WORD */
405   2                   CALL CLEARDATA(.CSBEGDATA,.CSENDDATA);
                            /* CLEAR VARIABLE DATA */
406   2                        INX,CURSORINP,CURSORSAVE = 1 ;
407   2                   ROLLSCREEN = TRUE;
408   2                   INACTMOD = 0FFH;
409   2                   INPREQUEST = FALSE;
410   2                   USART$OUT = ENTERPRIOR(.CSOUTPUT);
411   2                   USART$IN = ENTERPRIOR(.CSINPUT);
                            /* ENTER PRIORITY CALLS */
412   2                   INTMASK = INTMASK XOR 00001000B; /* SET INT 3 */
413   2                   INTMASK = INTMASK XOR 00010000B; /* SET INT 4 */
414   2                   RETURN;
415   2                   END SEIOSTART;

416   1         END;
```

```
MODULE INFORMATION:

     CODE AREA SIZE     = 047DH    1149D
     VARIABLE AREA SIZE = 017CH     380D
     MAXIMUM STACK SIZE = 000AH      10D
     870 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE PAIOMOD
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:PAIO.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            PA$IO$MOD:        DO;
                $NOLIST
                /*
                ****************************************************************
                **
                                PARALLEL IO DATA
                **
                ****************************************************************
                **
                **                      SUBSTITUTIONS
                */
 156   1        DCL      LPDATAOUT LIT '0E4H',       /* LP DATA OUTPUT PORT */
                         LPCMDOUT LIT '0E7H',        /* LP COMMAND OUTPUT PORT */
                         LPSTAT   LIT '0A0H',        /* MODE 1 OUTPUT */
                                  UP$STROBE  LIT '09H',      /* PORT C BIT 4 SET */
                                  DOWN$STROBE LIT '08H',     /* POST C BIT 4 RESET */
                         SET$INTA LIT '0DH',         /* PORT C BIT 6 SET */
                         LPOUTBUFLEN LIT '512',      /* OUTPUT BUFFER LENGTH */
                         FORMFEED LIT '0CH',
                         CR LIT '0DH',
                         LF LIT '0AH',
                         ENDSUBST LIT '0';
                /*
                ****************************************************************
                **
                **                      VARIABLE DATA
                */
 157   1        DCL      LPCLEARREG BYTE,
                             /* BEGIN OF CLEAR REGION */
                                 LPMN26 (4) BYTE,
```

```
        LPMN28 (4) BYTE,
        PRELL$OUT BYTE PUBLIC, /* PRIORITY INDEX */
        INACTMOD BYTE,         /* ACTUAL USER ID */
LPOUTBUF(LPOUTBUFLEN) BYTE,
    /* LP OUTPUT BUFFER */
(LPOUTX,LPOUTPTR) BYTE,
    /* INDICES FOR LP OUTPUT BUFFER
       LPOUTX - NEXT TO FILL
       LPOUTPTR - NEXT TO PRINT */
LPTELLBACK BYTE,
    /* TRUE IF TELLBACK MSG REQUESTED AFTER OUTPUT */
LPTBCODE BYTE,
    /* REQUESTED CODE FOR TELLBACK MSG */
LPOUTACTIVE BYTE,
    /* TRUE IF OUTPUT ACTIVE */
LPMODSAVE BYTE,
    /* MODULE NUMBER OF PRINTING MODULE */
LPCLEAREND BYTE;
```

```
                $ EJECT
                /*
                ***********************************************************************
                ***********************************************************************
                **
                                                                        LPOUTPUT
                    CONTINUE LINE PRINTER OUTPUT
                    PRIORITY PROCEDURE , CALL SCHEDULED BY INT 5
                **
                ***********************************************************************
                */
158   1         LP$OUTPUT:  PROCEDURE;

159   2                 IF LPOUTPTR >= LPOUTX THEN
                            /* NO MORE CHARACTERS TO PRINT */
160   2                   DO;
161   3                   LPOUTPTR = 0;LPOUTX = 0;
163   3                   LPOUTACTIVE = FALSE;
164   3                   IF LPTELLBACK THEN
                              /* TELLBACK MSG REQUESTED */
165   3                     DO;
166   4                     LPTELLBACK = FALSE;
167   4                     LPMN26(RMN) = LPMODSAVE;
168   4                     IF NOT SEND(.LPMN26(0)) THEN RETURN;
170   4                     MSGDATA(0) = LPTBCODE;
                                /* SEND TELLBACK MSG WITH REQUESTED CODE */
171   4                     END;
172   3                   RETURN;
173   3                   END;
174   2                         OUTPUT(LPCMDOUT) = SET$INTA ;
175   2                 OUTPUT(LPDATAOUT) = NOT LPOUTBUF(LPOUTPTR);
176   2                         OUTPUT(LPCMDOUT) = DOWN$STROBE ;
177   2                         OUTPUT(LPCMDOUT) = UP$STROBE ;
                            /* OUTPUT NEXT CHARACTER */
178   2                 LPOUTPTR = LPOUTPTR + 1;
179   2                 RETURN;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                LPSTARTOUT

                    START LINE PRINTER OUTPUT
                **
                ***************************************************************
                */
181    1        LPSTARTOUT: PROCEDURE;

182    2                IF LPOUTACTIVE THEN RETURN;
184    2                CALL LPOUTPUT;
185    2                LPOUTACTIVE = TRUE;
186    2                RETURN;
187    2                END LPSTARTOUT;
```

```
                $EJECT
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                        LPPACK

                    PACK 1 CHARACTER INTO OUTPUT BUFFER
                **
                **********************************************************************
                */
188   1         LPPACK: PROCEDURE (CHAR) BYTE;

189   2         DCL     CHAR BYTE;

190   2                 IF LPOUTX > LAST(LPOUTBUF)
                        THEN DO;
                             /* BUFFER OVERFLOW */
192   3                   CALL ERROR(STACKPTR,5,0);
193   3                   RETURN FALSE;
194   3                   END;
195   2                 ELSE DO;
                             /* PACK CHARACTER */
196   3                   LPOUTBUF(LPOUTX) = CHAR;
197   3                   LPOUTX = LPOUTX + 1;
198   3                   END;
199   2                 RETURN TRUE;
200   2                 END LPPACK;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                LPNEWLINE

                    POSITION PRINT HEAD AT BEGIN OF NEW LINE
                **
                ***************************************************************
                */
201     1       LPNEWLINE:  PROCEDURE;

202     2               IF NOT LPPACK(CR) THEN RETURN;
204     2               IF NOT LPPACK(LF) THEN RETURN;
206     2               CALL LPSTARTOUT;
207     2               RETURN;
208     2               END LPNEWLINE;
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                PANEWPAGE

                    POSITION PRINT HEAD AT TOP OF NEW PAGE
                **
                ***************************************************************
                */
209     1       PA$NEWPAGE: PROCEDURE PUBLIC ;

210     2               IF NOT LPPACK(FORMFEED) THEN RETURN;
212     2               CALL LPSTARTOUT;
213     2               RETURN;
214     2               END PANEWPAGE;
```

```
$EJECT
/*
**********************************************************************
**********************************************************************
**
                                                              PAPRINT

    PROCESS PRINT MSG     (MN 23)
    IF MSGDATA(0) = TRUE : PRINT HEAD AT BEGIN OF NEW LINE
**
**********************************************************************
*/
215   1       PA$PRINT:   PROCEDURE PUBLIC ;

216   2                       IF MSG(SMN) <> INACTMOD THEN RETURN;
218   2               B3 = MSG(ML) - 5;
219   2               IF B3 = 0 THEN RETURN;
221   2               DO B2 = 1 TO B3;
                          /* MOVE CHARACTERS INTO OUTPUT BUFFER */
222   3                 IF NOT LPPACK(MSGDATA(B2)) THEN GO TO PR1;
                            /* OUTPUT BUFFER FULL */
224   3                 END;
225   2               IF MSGDATA(0) THEN CALL LPNEWLINE;
227   2       PR1:        ;
228   2               CALL LPSTARTOUT;
229   2               RETURN;
230   2               END PAPRINT;
```

```
$EJECT
/*
*****************************************************************
*****************************************************************
**
                                                    PAPRINTLINKED

    PROCESS PRINT MSG WITH TELLBACK    (MN 25)
**
*****************************************************************
*/
231   1   PA$PRINT$LINKED:    PROCEDURE PUBLIC ;

232   2                   IF MSG(SMN) <> INACTMOD THEN RETURN;
234   2             IF MSG(ML) < 7 THEN RETURN;
236   2             LPTBCODE = MSGDATA(0);
                        /* SAVE TELLBACK CODE */
237   2             LPTELLBACK = TRUE;
238   2             ADRMSGDATA = ADRMSGDATA + 1;
239   2             MSG(ML) = MSG(ML) - 1;
                        /* ADJUST MSG FOR PROCESS BY LPPRINT */
240   2             LPMODSAVE = MSG(SMN);
                        /* SAVE MODULE NUMBER FOR TELLBACK MSG */
241   2             CALL PAPRINT;
242   2             RETURN;
243   2             END PAPRINTLINKED;
```

```
                $EJECT
                /*
                ****************************************************************
                ****************************************************************
                **
                                                        PA$OUT$ACTIVATE
                : CONNECT SENDER WITH PARALLEL DEVICE
                ****************************************************************
                */
  244   1       PA$OUT$ACTIVATE: PROCEDURE PUBLIC;
  245   2               LPMN28(0) = MSG(SMN);
  246   2               IF NOT SEND(.LPMN28) THEN RETURN;
  248   2               IF INACTMOD <> 0FFH
                           THEN MSGDATA(0) = FALSE;
  250   2                   ELSE DO;
  251   3                        MSGDATA(0) = TRUE;
  252   3                        INACTMOD = MSG(SMN);
  253   3                        END;
  254   2               RETURN;
  255   2               END;
                /*
                ****************************************************************
                ****************************************************************
                **
                                                        PA$OUT$RELEASE
                : DISCONNECT SENDER FROM PARALLEL DEVICE
                ****************************************************************
                */
  256   1       PA$OUT$RELEASE: PROCEDURE PUBLIC ;

  257   2                INACTMOD = 0FFH ;
  258   2                 RETURN;
  259   2                END;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                PAIO$START

                    INITIALIZE PARALLEL IO MODULE
                **
                ***************************************************************
                */
 260    1       PAIO$START: PROCEDURE PUBLIC;

 261    2               CALL CLEARDATA(.LPCLEARBEG,.LPCLEAREND);
                             /* CLEAR VARIABLE DATA */
 262    2                   LPMN26(1)=EX;LPMN26(2)=26;LPMN26(3)=5;
 265    2                   LPMN28(1)=EX;LPMN28(2)=28;LPMN28(3)=5;
 268    2                   PRELL$OUT = ENTERPRIOR(.LPOUTPUT);
 269    2                   INACTMOD = 0FFH ;
 270    2                   OUTPUT(LPCMDOUT) = LPSTAT; /* MODE 1 OUTPUT */
 271    2                   OUTPUT(LPCMDOUT) = UP$STROBE; /* INIT STROBE LEVEL UP */
 272    2               INTMASK = INTMASK XOR 00100000B; /* SET INT 5 */
 273    2               RETURN;
 274    2               END PAIO$START;
 275    1       END;
```

158

MODULE INFORMATION:

```
     CODE AREA SIZE      = 01D6H    470D
     VARIABLE AREA SIZE  = 0213H    531D
     MAXIMUM STACK SIZE  = 000AH     10D
     499 LINES READ
     0 PROGRAM ERROR(S)
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE MONITOR
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:MONI.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            MONITOR: DO;
                $NOLIST
                /*
                ************** CONSTANT DATA **********************
                */
 174    1       DCL EOT LIT '04H'; /* END OF TEXT ID */

 175    1       DCL COMMAND$LIST(8) BYTE DATA('C','D','F','G','M','S','T','P'),
                    MONITOR$TAG(18) BYTE DATA(1,'MONITOR REQUEST ',CPTR+30H),
                    MONITOR$IN (11) BYTE DATA(1,'MONITOR IN'),
                    MONITOR$OUT(11) BYTE DATA('MONITOR OUT'),
                    MON$PROMT (2) BYTE DATA(0,'.');
                /*
                ************** VARIABLE DATA **********************
                */
 176    1       DCL COMMAND BYTE ,          /* MONITOR COMMAND VALUE */
                    (ADR1,ADR2,ADR3) ADDRESS, /* MONITOR PARAMETTERS */
                    (LOWA,HIGHA) BYTE AT(.ADR1),/* OVERLAP OF ADR1 */
                    (BYTE1 BASED ADR1) BYTE,
                    SUBS$IN$CHARGE BYTE,  /* SUBSTITUTE COMMAND FLAG */
                    SAVED$FLAG BYTE,
                    SAVED$INSTRUCTION BYTE,
                    SAVED$LOCATION ADDRESS, /* DATA FOR MONITOR TRAP */
                    END$MON$DATA BYTE;
```

```
                $EJECT
                /*
                ***************************************************
                : PROMTS CHARACTER "."
                   AND REQUEST AND INPUT MESSAGE STRING FROM
                   THE SYSTEM CONSOLE.
                ***************************************************
                */
177   1         RECEIVE$NEXT$COMMAND: PROCEDURE;

178   2                   CALL PRINT$SYNC(.MON$PROMT,2,EX);
179   2                   CALL CON$INPUT$REQ(EX,TRUE);
180   2                   END;

                /*
                ***************************************************
                 : EXTRACTS COMMAND'S PARAMETERS FROM
                   MESSAGE INPUT.
                ***************************************************
                */
181   1         GET$PARAMETERS: PROCEDURE (P1) BYTE;
182   2                   DCL P1 BYTE, A ADDRESS,
                            (ADR BASED A) ADDRESS;

183   2                   A = .ADR1 ;
184   2                   DO WHILE P1 <> 0 ;
185   3                      IF NOT GETNUM THEN DO;
187   4                                         CALL RECEIVE$NEXT$COMMAND;
188   4                                         RETURN FALSE;
189   4                                         END;
190   3                      ADR = A4 ;
191   3                      A = A + 2 ; P1 = P1 - 1 ;
193   3                      END;
194   2                   RETURN TRUE ;
195   2                   END;
```

160

```
                $EJECT
                /*
                ***************************************************
                 : DUMPS A 16-CHARACTER LINE FROM MEMORY TO
                   THE SYSTEM CONSOLE. ADR1 POINTS TO THE BEGGINING
                   OF THE LINE. ENDS WHEN IT GETS TO ADR2.
                ***************************************************
                */
196   1         PRINT$LINE: PROCEDURE PUBLIC;
197   2                     DCL LINE (56) BYTE;

198   2                     CALL FILL(.LINE,.LINE+55,' ');/* CLEAR LINE */
199   2                     LINE(0) = 1 ; /* CRLF AFTER PRINT */
200   2                     CALL NUMOUT(ADR1,16,'0',.LINE(1),4);
201   2                     LINE(5) = 'H' ; LINE(6) = ':' ; /* ADDRESS SET */
203   2                     B2 = 8 ;
204   2                     DO WHILE ADR1 <= ADR2 AND B2 < 55 ;
205   3                        CALL CONVASC(BYTE1);
206   3                        LINE(B2) = BU ; LINE(B2+1) = BL ;
208   3                        B2 = B2 + 3 ; ADR1 = ADR1 + 1 ;
210   3                        END;
211   2                     IF ADR1 > ADR2 THEN DO ;
213   3                                         CALL PRINT$SYNC(.LINE,56,EX);
214   3                                         CALL RECEIVE$NEXT$COMMAND;
215   3                                         END;
216   2                                    ELSE CALL PRINT$LINKED(.LINE,56,EX,0);
217   2                     RETURN;
218   2                     END;
```

```
                $EJECT
                /*
                ***************************************************************
                 : DISCONNECTS CURRENT MONITOR FROM THE SYSTEM CONSOLE
                   AND REQUEST CONNECTION OF THE INDICATED BY THE
                   INPUTED PARAMETER.
                ***************************************************************
                */
219   1         CHANGE$COMPUTER: PROCEDURE;
220   2                   DCL COMPUTER BYTE;

221   2                   IF NOT GET$PARAMETERS(1) THEN RETURN;
223   2                   IF LOWA < 1 OR LOWA > 3 THEN CALL RECEIVE$NEXT$COMMAND;
225   2                                 ELSE DO;
226   3                                      START(LOWA-1) = LOWA;
227   3                                      COMPUTER = SHL((LOWA-1),3); /* TIMES 8 */
228   3                                      CALL CON$RELEASE ;
229   3                                      CALL CON$LINK$REQ(COMPUTER);
230   3                                      END;
231   2                   RETURN;
232   2                   END;
                /*
                ***************************************************************
                 : FIILS OUT MEMORY BOUNDED BY ADR1 AND ADR2
                   WITH CHARACTER INDICATED WITH THIRD PARAMETER.
                ***************************************************************
                */
233   1         FILLER: PROCEDURE;
234   2                   IF NOT GET$PARAMETERS(3) THEN RETURN;
236   2               B2 = LOW(ADR3);
237   2                   CALL FILL(ADR1,ADR2,B2);
238   2               CALL RECEIVE$NEXT$COMMAND;
239   2                   RETURN;
240   2                   END;
```

```
                $EJECT
                /*
                :*************************************************************
                 : DISPLAYS MEMORY BOUNDED BY ADR1 AND ADR2
                   ON SYSTEM CONSOLE.
                *************************************************************
                */
241    1        DISPLAY: PROCEDURE;
242    2                 IF NOT GET$PARAMETERS(2) THEN RETURN;
244    2                 IF ADR1 > ADR2 THEN DO;
246    3                                     CALL RECEIVE$NEXT$COMMAND;
247    3                                     RETURN;
248    3                                     END;
249    2                 CALL PRINT$LINE ;
250    2                 RETURN;
251    2                 END;
                /*
                *************************************************************
                 : MOVES MEMORY BLOCK BOUNDED BY ADR1 AND ADR2
                   TO LOCATION POINTED BY ADR3.
                *************************************************************
                */
252    1        MOVE$MEMORY: PROCEDURE;
253    2                 IF NOT GET$PARAMETERS(3) THEN RETURN;
255    2                 IF ADR1 > ADR2 OR (ADR2>=ADR3 AND ADR3>=ADR1) THEN DO;
257    3                                               CALL RECEIVE$NEXT$COMMAND;
258    3                                               RETURN;
259    3                                               END;
260    2                 CALL MOVE(ADR2-ADR1+1,ADR1,ADR3);
261    2                 CALL RECEIVE$NEXT$COMMAND;
262    2                 RETURN;
263    2                 END;
```

```
                $EJECT
                /*
                ***************************************************************
                 : GETS A CHARACTER FROM MEMORY AND
                   SENDS IT TO THE SYSTEM CONSOLE.
                ***************************************************************
                */
264    1        SEND$MEM: PROCEDURE;
265    2            DCL MEMDAT (5) BYTE;

266    2            MEMDAT(0) = 0 ;/* NOT ROLL */
267    2            MEMDAT(1) = ' ';
268    2            CALL CONVASC(BYTE1); /* SET FOR OUTPUT */
269    2            MEMDAT(2) = BU;
270    2            MEMDAT(3) = BL;
271    2            MEMDAT(4) = '-';
272    2            CALL PRINT$SYNC(.MEMDAT,5,EX);/* OUTPUT */
273    2            CALL CON$INPUT$REQ(EX,FALSE);
274    2            RETURN;
275    2            END;
                /*
                ***************************************************************
                 : SUBSTITUTE A MEMEORY BYTE
                ***************************************************************
                */
276    1        SUBS$NEXT: PROCEDURE;
277    2                B1 = B1 - 1; /* COME BACK TO LAST CHARACTER */
278    2            IF NOT GETNUM THEN DO;
280    3                    SUBS$IN$CHARGE = FALSE;
281    3                    CALL RECEIVE$NEXT$COMMAND;
282    3                    RETURN; END;
284    2            BYTE1 = BL;
285    2            ADR1 = ADR1 + 1 ;
286    2            CALL SEND$MEM;
287    2            RETURN;
288    2            END;
```

```
                $EJECT
                /*
                ***************************************************
                 : TRIGGERS THE SUBSTITUTION SEQUENCE.
                   A CHARACTER FROM MEMORY WILL BE DISPLAYED
                   AND ANOTHER COMMING FROM THE
                   SYSTEM CONSOLE PUT IN ITS PLACE.
                ***************************************************
                */
289   1         SUBSTITUTE: PROCEDURE;
290   2             IF NOT GET$PARAMETERS(1) THEN RETURN;
292   2             SUBS$IN$CHARGE = TRUE;
293   2             CALL SEND$MEM;
294   2             RETURN;
295   2             END;
                /*
                ***************************************************
                 : DISCONNECTS THE MONITOR FROM THE SYSTEM CONSOLE.
                   OPTIONALLY, IT PUTS A MONITOR TRAP
                   INTO AN INDICATED MEMORY ADDRESS.
                ***************************************************
                */
296   1         GOING: PROCEDURE;
297   2                  IF GETNUM THEN DO;
299   3                                 SAVED$FLAG = TRUE ;
300   3                                 SAVED$LOCATION,ADR1 = A4 ;
301   3                                 SAVED$INSTRUCTION = BYTE1;
302   3                                 BYTE1 = 11001111B; /* RST 1 */
303   3                                 CALL ENTER$INT(1);
304   3                                 END;
305   2                  CALL CON$RELEASE;
306   2                  CALL PRINT$ASYNC(.MONITOR$OUT,11);
307   2                  RETURN;
308   2                  END;
```

```
                $EJECT
                /*
                ***************************************************
                 : TRANSMIT A MESSAGE COMMING FROM THE SYSTEM CONSOLE.
                   THE MESSAGE IS SPECIFIED BY A RECEIVER NUMBER
                   AND A MESSAGE NUMBER.
                ***************************************************
                */
309   1         TRANSMIT: PROCEDURE;
310   2                   DCL TMSG(4) BYTE;

311   2                   IF NOT GET$PARAMETERS(2) THEN RETURN;
313   2                   TMSG(1) = EX ; TMSG(3) = 4 ;
315   2                   TMSG(0) = LOWA ; TMSG(2) = LOW(ADR2) ;
317   2                   IF NOT SEND(.TMSG) THEN DO;
319   3                                        CALL RECEIVE$NEXT$COMMAND;
320   3                                        RETURN; END;
322   2                CALL CON$RELEASE;
323   2                CALL PRINT$ASYNC(.MONITOR$OUT,11);
324   2                   RETURN;
325   2                   END;
                /***************************************************
                 : CHANGES THE SYSTEM PRINTER TO ANOTHER COMPUTER.
                ***************************************************
                */
326   1         CHANGE$PRINTER: PROCEDURE;
327   2             DCL PRINTER BYTE;

328   2             IF NOT GET$PARAMETERS(1) THEN RETURN;
330   2             IF LOWA > 0  AND  LOWA < 4 THEN /* CORRECT INPUT */
331   2                 DO;PRINTER = SHL((LOWA-1),3);
333   3                    SYSMN14(0),SYSMN19(0),SYSMN23(0),
                           SYSMN24(0),SYSMN25(0) = PRINTER;
334   3                    END;
335   2             CALL RECEIVE$NEXT$COMMAND;
336   2             RETURN;
```

```
                $EJECT
                /*
                ***********************************************************
                 : USED TO RESTORE ORIGINAL MEMORY VALUES
                   WHEN MONITOR TRAP.
                ***********************************************************
                */
338   1         RESET$INSTRU: PROCEDURE;
339   2                       DCL (P1 BASED SAVED$LOCATION) BYTE;

340   2                       P1 = SAVED$INSTRUCTION;
341   2                       SAVED$FLAG = FALSE;
342   2                       CALL REM$INT(1);
343   2                       RETURN;
344   2                       END;
                /*
                ***********************************************************
                 : EXECUTED BY INTERRUPT HANDLER NUMBER 1, USED FOR
                   TRAPPING THE MONITOR IN A MSPECIFIC MEMORY LOCATION.
                ***********************************************************
                */
345   1         MONITOR$TRAP: PROCEDURE PUBLIC;

346   2                 IF SAVED$FLAG THEN CALL RESET$INSTRU;
348   2                 CALL MONITOR;
349   2                 RETURN;
350   2                 END;
```

```
                $EJECT
                /*****************************************************
                ******************************************************
                 : ALL MONITOR COMMANDS COMMING FROM THE SYSTEM CONSOLE
                   WILL GET TO THIS PROCEDURE. ONCE THE COMMAND
                   IS RECOGNIZED THE CONTROL WILL BE CHANGED TO THE
                   CORRESPONDING PROCEDURE.
                ******************************************************/
  351    1      MONI$INTER: PROCEDURE PUBLIC;
  352    2          B1 = 0 ; /* SET TO BEGINNING OF INPUT STRING */
  353    2          IF NOT GET$CHAR THEN DO;
  355    3                   CALL RECEIVE$NEXT$COMMAND;
  356    3                   SUBS$IN$CHARGE = FALSE;
  357    3                   RETURN; END;
  359    2              IF SUBS$IN$CHARGE THEN DO;
  361    3                                     CALL SUBS$NEXT;
  362    3                                     RETURN;
  363    3                                     END;
  364    2              COMMAND = BL;
  365    2              DO B2 = 0 TO LAST(COMMAND$LIST);
  366    3                 IF COMMAND$LIST(B2) = COMMAND
                                  THEN GOTO INTER$COMMAND; END;
  369    2              CALL RECEIVE$NEXT$COMMAND;
  370    2              RETURN;
  371    2              INTER$COMMAND: ; DO CASE B2;
  373    3                    CALL CHANGE$COMPUTER;
  374    3                    CALL DISPLAY ;
  375    3                    CALL FILLER  ;
  376    3                    CALL GOING ;
  377    3                    CALL MOVE$MEMORY;
  378    3                    CALL SUBSTITUTE;
  379    3                    CALL TRANSMIT;
  380    3                    CALL CHANGE$PRINTER;
  381    3                    END;
  382    2                RETURN;
  383    2              END;
```

```
                $EJECT
                /****
                **************************************************************
                 : CHECK IF SYSTEM CONSOLE CONNECTION HAS BEEN ACCEPTED,
                   AND REQUEST THE FIRST COMMAND FROM THERE.
                **************************************************************
                */
 384    1       MONI$INIT: PROCEDURE PUBLIC;
 385    2                  IF NOT MSG$DATA(0) THEN
 386    2                         CALL PRINT$ASYNC(.MONITOR$TAG,18);
 387    2                     ELSE CALL PRINT$SYNC(.MONITOR$IN,11,EX);
 388    2                  CALL RECEIVE$NEXT$COMMAND;
 389    2                  RETURN;
 390    2                  END;
                /*
                **************************************************************
                 : RECEIVES DISCONNECTION MESSAGE FROM SYSTEM CONSOLE
                   AND ENDS THE MONITOR INPUT COMMAND SEQUENCE.
                **************************************************************
                */
 391    1       MONI$ENDER: PROCEDURE PUBLIC;
 392    2                   IF SAVED$FLAG THEN CALL RESET$INSTRU;
 394    2                   CALL PRINT$ASYNC(.MONITOR$OUT,11);
 395    2                   RETURN;
 396    2                   END;


 397    1       END;



MODULE INFORMATION:

     CODE AREA SIZE     = 042AH    1066D
     VARIABLE AREA SIZE = 0053H      83D
     MAXIMUM STACK SIZE = 000CH      12D
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB1
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SC1.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


  1             SCPUB1:    DO;
                $ NOLIST

 26    1        ERROR: PROCEDURE(P1,P2,P3) EXTERNAL;
 27    2                 DCL P1 ADDRESS,(P2,P3) BYTE;
 28    2                 END;

                /*
                ***********************************************************************
                :**********************************************************************
                **
                                                                              FILL
                    FILLS MEMORY WITH SPECIFIED CHARACTER
                        INPUT : P1 - ADDRESS LOWER BOUND
                                P2 - ADDRESS UPPER BOUND
                                P3 - FILLING CHARACTER
                **
                ***********************************************************************
                */
 29    1        FILL: PROCEDURE (P1,P2,P3) PUBLIC;
 30    2              DCL (P1,P2) ADDRESS,P3 BYTE,
                        (X BASED P1) BYTE;

 31    2              DO WHILE P1 <= P2 ;
 32    3                 X = P3 ;
 33    3                 P1 = P1 + 1 ;
 34    3                 END;

 35    2              RETURN;
 36    2              END FILL;
```

```
                $EJECT
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                        CLEARDATA

                    CLEAR MODULES VARIABLE DATA REGION
                        INPUT : P1 - ADDRESS OF FIRST BYTE
                                P2 - ADDRESS OF LAST BYTE
                **
                **********************************************************************
                */
 37   1         CLEARDATA:   PROCEDURE (P1,P2) PUBLIC;

 38   2         DCL            (P1,P2) ADDRESS;

 39   2                           CALL FILL(P1,P2,0);
 40   2                       RETURN;
 41   2                       END CLEARDATA;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                        SHFT

                    RETURN BYTE WITH A 1 SHIFTED INTO POSITION P1
                        INPUT  : P1 - BIT POSITION
                        OUTPUT : P1TH POWER OF 2
                **
                ***************************************************************
                */
  42    1       SHFT:       PROCEDURE (P1) BYTE PUBLIC;

  43    2       DCL         P1 BYTE;

  44    2                   IF P1 = 0 THEN RETURN 1;
  46    2                   RETURN ROL(1,P1);
  47    2                   END;
```

```
                $EJECT
                /*
                ****************************************************************
                ****************************************************************
                **
                                                                        BIT

                        CHECK A BIT IN A SPECIFIED BYTE
                        INPUT  : P1 - BIT
                                 P2 - BYTE TO CHECK
                        OUTPUT : TRUE IF BIT IS SET,FALSE IF NOT
                **
                ****************************************************************
                */
48   1          BIT:        PROCEDURE (P1,P2) BYTE PUBLIC;

49   2          DCL         (P1,P2) BYTE;

50   2                      IF (P2 AND SHFT(P1)) = 0 THEN RETURN FALSE;
                                /* SPECIFIED BIT NOT SET */
52   2                      RETURN TRUE;
53   2                      END BIT;
```

```
$EJECT
/*
**********************************************************************
**********************************************************************
**
                                                              CLEARBIT

    CLEAR A BIT IN A SPECIFIED BYTE
        INPUT : P1 - BIT
                P2 - ADDRESS OF BYTE
**
**********************************************************************
*/
54   1      CLEARBIT:   PROCEDURE (P1,P2) PUBLIC;

55   2      DCL      P1 BYTE,P2 ADDRESS,
                     (WORD BASED P2) BYTE;

56   2                   WORD = WORD AND (NOT SHFT(P1)) ;
                         /* CLEAR BIT */
57   2                  RETURN;
58   2                  END CLEARBIT;
```

```
$EJECT
/*
**********************************************************************
**********************************************************************
**
                                                              SETBIT

    SET A BIT IN A SPECIFIED BYTE
        INPUT : P1 - BIT
                P2 - ADDRESS OF BYTE
**
**********************************************************************
*/
59   1      SETBIT:     PROCEDURE (P1,P2) PUBLIC;

60   2      DCL         P1 BYTE,P2 ADDRESS,
                        (WORD BASED P2) BYTE;

61   2                  WORD = WORD OR SHFT(P1);
                            /* SET BIT */
62   2                  RETURN;
63   2                  END SETBIT;
```

```
                $EJECT
                /*
                ****************************************************************
                ****************************************************************
                **
                                                                        LEGAL

                    CHECK P1 AGAINST P2
                        INPUT  : P1,P2 VALUES TO CHECK
                                 P3 ERROR CODE FOR SYSMON
                        OUTPUT : TRUE IF P1 < P2
                                 FALSE OTHERWISE
                **
                ****************************************************************
                */
  64   1        LEGAL:      PROCEDURE (P1,P2,P3) BYTE PUBLIC;

  65   2        DCL         (P1,P2,P3) BYTE;

  66   2                    IF P1 < P2 THEN RETURN TRUE;
  68   2                    CALL ERROR(STACKPTR,0,P3);
  69   2                    RETURN FALSE;
  70   2                    END LEGAL;
  71   1        END SCPUB1;


MODULE INFORMATION:

     CODE AREA SIZE     = 00D3H    211D
     VARIABLE AREA SIZE = 0015H     21D
     MAXIMUM STACK SIZE = 0006H      6D
     301 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB2
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SC2.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1          SCPUB2:     DO;

              $NOLIST

  26   1      LEGAL: PROCEDURE (P1,P2,P3) BYTE EXTERNAL;
  27   2             DCL (P1,P2,P3) BYTE;
  28   2             END;
  29   1      GET: PROCEDURE (P1) EXTERNAL;
  30   2           DCL P1 ADDRESS;
  31   2           END;
  32   1      RELEASE: PROCEDURE (P1) EXTERNAL;
  33   2                DCL P1 ADDRESS;
  34   2                END;
  35   1      ERROR: PROCEDURE(P1,P2,P3) EXTERNAL;
  36   2             DCL P1 ADDRESS,(P2,P3) BYTE;
  37   2             END;
  38   1      VECTOR$ADD: PROCEDURE(P1,P2,P3) EXTERNAL;
  39   2                  DCL (P1,P2,P3) ADDRESS;
  40   2                  END;
```

```
                $EJECT
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                        COPY$CLOCK
                COPY REAL TIME CLOCK INTO
                MEMORY POINTED BY P1
                **
                **********************************************************************
                */
  41    1       COPY$CLOCK: PROCEDURE(P1) PUBLIC;
  42    2               DCL P1 ADDRESS;

  43    2               CALL GET(.CLOCK);
  44    2               CALL MOVE(4,.RTC,P1);/* COPY REAL TIME */
  45    2               CALL RELEASE(.CLOCK);

  46    2               RETURN;
  47    2               END COPY$CLOCK;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                SETPERTIME

                    SET NEXT CALL TIME FOR PERIODIC
                        INPUT: P1 - PERIODIC INDEX
                **
                ***************************************************************
                */
 48    1        SETPERTIME: PROCEDURE (P1) PUBLIC;

 49    2        DCL     (P1,CY) BYTE, TIME(4) BYTE;

 50    2                XA1 = PERLIST(P1).PERINTADR ;
 51    2                  B3 = PERINT(4) ;
 52    2               XA2 = .PERLIST(P1).PERTIME;

 53    2                CALL COPY$CLOCK(.TIME);
 54    2                CALL VECTOR$ADD(.TIME,XA1,XA2);

 55    2               RETURN;
 56    2               END SETPERTIME;
```

```
                $EJECT
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                        TIME$CMP

                        COMPARE TWO CLOCKS:
                            INPUT  : P1 ADDRESS OF FIRST CLOCK VALUE
                                     P2 ADDRESS OF SECOND CLOCK VALUE
                            OUTPUT : TRUE IF (P1) < (P2)
                                     FALSE OTHERWISE
                **
                **********************************************************************
                */
  57    1       TIME$CMP: PROCEDURE(P1,P2) BYTE PUBLIC;

  58    2       DCL         (P1,P2) ADDRESS,
                            (TIM1 BASED P1) (4) BYTE,
                            (TIM2 BASED P2) (4) BYTE;

  59    2                   DO XB3 = 0 TO 3 ;
  60    3                     IF TIM1(XB3) < TIM2(XB3) THEN RETURN TRUE;
  62    3                     IF TIM1(XB3) > TIM2(XB3) THEN RETURN FALSE;
  64    3                     END;

  65    2                   RETURN FALSE; /* EQUALS */
  66    2                 END TIME$CMP;
```

```
                $EJECT
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                        PERACT

                    ACTIVATE PERIODIC CALL OF A PROCEDURE
                        INPUT  : P1 - ADDRESS OF PROCEDURE
                                 P2 - ADDRESS OF TIME INTERVAL(RTC FORMAT)
                        OUTPUT : INDEX OF PERIODIC
                **
                **********************************************************************
                */
 67   1     PERACT:     PROCEDURE (P1,P2) BYTE PUBLIC;

 68   2     DCL         (P1,P2) ADDRESS, TIME(4) BYTE;

 69   2                 DO XB1 = 0 TO LAST(PERLIST);
                           /* SEARCH FOR FREE SLOT */
 70   3                   IF PERLIST(XB1).FREE THEN GO TO PERACT1;
                             /* FREE SLOT FOUND */
 72   3                   END;
 73   2                 CALL ERROR(.PERACT,0,3);
 74   2                 RETURN 0FFH;
 75   2     PERACT1:    PERXTBL(NUMPER) = XB1;
 76   2                 PERLIST(XB1).FREE = FALSE;
 77   2                 PERLIST(XB1).PERADR = P1;
 78   2                 PERLIST(XB1).PERINTADR = P2;
 79   2                     CALL COPY$CLOCK(.TIME);
 80   2                     CALL VECTOR$ADD(.TIME,P2,.PERLIST(XB1).PERTIME);
                           /* SET NEXT CALL TIME */
 81   2                 NUMPER = NUMPER + 1;
 82   2                 RETURN PERXTBL(NUMPER-1);
 83   2                 END PERACT;
```

```
                $EJECT
                /*
                ***********************************************************
                ***********************************************************
                **
                                                                    PERCHG

                    CHANGE THE TIME INTERVAL OF A PERIODIC CALL
                        INPUT: P1 - PERIODIC INDEX
                               P2 - ADDRESS OF NEW TIME INTERVAL
                **
                ***********************************************************
                */
 84    1        PERCHG:     PROCEDURE (P1,P2) PUBLIC;

 85    2        DCL      P1 BYTE,P2 ADDRESS;

 86    2                 IF NOT LEGAL(P1,MAXPER,4) THEN RETURN;
                             /* ILLEGAL PERIODIC INDEX */
 88    2                 PERLIST(P1).PERINTADR = P2;
                         /* SET NEW TIME INTERVAL */
 89    2                 CALL SETPERTIME(P1);
                         /* SET NEW CALL TIME */
 90    2                 RETURN;
 91    2                 END PERCHG;
```

```
                $EJECT
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                        PERSUSP

                    SUSPEND THE PERIODIC CALL OF A PROCEDURE
                        INPUT: PERIODIC INDEX
                **
                **********************************************************************
                */
 92    1        PERSUSP:    PROCEDURE (P1) PUBLIC;

 93    2        DCL         P1 BYTE;

 94    2                    IF NOT LEGAL(P1,MAXPER,4) THEN RETURN;
                                /* ILLEGAL PERIODIC INDEX */
 96    2                    PERLIST(P1).FREE = TRUE;
 97    2                    NUMPER = NUMPER - 1;
 98    2                    IF NUMPER = 0 THEN GO TO SUSP1;
100    2                    XB2 = FALSE;
101    2                    DO XB1 = 0 TO NUMPER;
                                /* REMOVE P1 FROM PERXTBL AND COMPACT PERXTBL */
102    3                      IF NOT XB2 AND PERXTBL(XB1) = P1 THEN XB2 = TRUE;
104    3                      IF XB2 THEN PERXTBL(XB1) = PERXTBL(XB1+1);
106    3                      END;
107    2        SUSP1:         PERXTBL(NUMPER) = 0FFH;
108    2                    RETURN;
109    2                    END;
110    1        END SCPUB2;
```

183

MODULE INFORMATION:

```
     CODE AREA SIZE      = 028DH    653D
     VARIABLE AREA SIZE  = 0018H     24D
     MAXIMUM STACK SIZE  = 0006H      6D
     325 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB3
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SC3.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            SCPUB3:     DO;
                $NOLIST

  26    1       ERROR:PROCEDURE (P1,P2,P3) EXTERNAL;
  27    2             DCL P1 ADDRESS,(P2,P3) BYTE;
  28    2            END;
  29    1       BIT: PROCEDURE (P1,P2) BYTE EXTERNAL;
  30    2            DCL (P1,P2) BYTE;
  31    2            END;
  32    1       PRINT$ASYNC: PROCEDURE (P1,P2) EXTERNAL;
  33    2                    DCL P1 ADDRESS, P2 BYTE;
  34    2                    END;
  35    1       CONVASC: PROCEDURE (P1) EXTERNAL;
  36    2                DCL P1 BYTE;
  37    2                END;
```

```
                $EJECT
                /*
                ****************************************************************
                ****************************************************************
                **
                                                                MSGOVERFLOW

                    HANDLE MSG OVERFLOW
                **
                ****************************************************************
                */
   38    1      MSGOVERFLOW:    PROCEDURE PUBLIC;

   39    2              BU = MSG(MN);
   40    2              BL = MSG(SMN);
   41    2              CODESAVE = A4;
   42    2                      CALL ERROR(CODESAVE,0,1);
   43    2              RETURN;
   44    2              END MSGOVERFLOW;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                    PACKMCB

                    PACK MCB INTO MSGBUFFER
                        INPUT: ADDRESS OF MCB
                **
                ***************************************************************
                */
45    1         PACKMCB:     PROCEDURE (A) PUBLIC;

46    2         DCL        A ADDRESS;

47    2                      CALL MOVE(4,A,.MSGBUFFER(MSGIN));
48    2                      MSGIN = MSGIN + 4;

49    2                      RETURN;
50    2                  END PACKMCB;
```

```
                $EJECT
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                          SEND

                    SEND A MESSAGE TO ANOTHER MODULE IN THE SYSTEM
                        INPUT  : ADDRESS OF MSG CONTROL BLOCK(MCB)
                                 MCB : RMN  * RECEIVING MODULE NUMBER
                                       SMN  * SENDING MODULE NUMBER
                                        MN  * MSG NUMBER
                                        ML  * MSG LENGTH
                        OUTPUT : FALSE IF RECEIVING MODULE IS NOT ACTIVATED
                                   OR DOES NOT EXIST OR RMN IS ILLEGAL
                                 TRUE OTHERWISE
                                 ADRMSGDATA IS SET TO ADDRESS OF FIRST DATA BYTE
                **
                **********************************************************************
                */
51    1         SEND:       PROCEDURE (P1) BYTE PUBLIC;

52    2         DCL         (P1) ADDRESS,SAVEMSGIN ADDRESS,
                          (MSG BASED P1) (4) BYTE;

53    2                   IF NOT BIT(1,MODSTATUS(MSG(RMN))) OR MSG(RMN) > MAXSYSMOD - 1
                          THEN RETURN FALSE;
                              /* RECEIVING MODULE NOT ACTIVATED OR DOES NOT
                                 EXIST */
55    2                   IF MSGOUT = MSGIN AND NUMMSG <> 0 THEN GO TO SEND2;
                              /* MSG BUFFER OVERFLOW */
57    2                   IF MSGOUT <= MSGIN THEN
58    2                     DO;
59    3                     IF (LAST(MSGBUFFER) - MSGIN) >= (MSG(ML) - 1) THEN
60    3                       GO TO SEND1;
                                /* MSG FITS INTO REST OF BUFFER */
```

```
 61   3                LASTMSG = MSGIN;
 62   3                MSGIN = 0;
 63   3                END;
 64   2              IF (MSGOUT - MSGIN) < MSG(ML) THEN GO TO SEND2;
                          /* NOT ENOUGH SPACE FOR MSG,OVERFLOW */
 66   2      SEND1:    SAVEMSGIN = MSGIN;
 67   2              CALL PACKMCB(P1);
                          /* TRANSFER MCB */
 68   2              MSGIN = SAVEMSGIN + MSG(ML);
                          /* COMPUTE BEGIN FOR NEXT MSG */
 69   2              IF MSGIN > LAST(MSGBUFFER) THEN
                          /* MSGIN IS OUTSIDE MSGBUFFER */
 70   2                DO;
 71   3                LASTMSG = MSGIN;
 72   3                MSGIN = 0;
 73   3                END;
 74   2              NUMMSG = NUMMSG + 1;
 75   2              ADRMSGDATA = .MSGBUFFER(SAVEMSGIN + 4);
                          /* SET ADDRESS OF FIRST DATA BYTE */
 76   2              RETURN TRUE;
 77   2      SEND2:      ADRMSG = P1;
 78   2              CALL MSGOVERFLOW;
 79   2              RETURN FALSE;
 80   2              END SEND;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                        GET

                        GET VARIABLE LOCK FOR USER
                    INPUT: VARIABLE POINTER.
                **
                ***************************************************************
                */
 81   1         GET:    PROCEDURE(KEY) PUBLIC;
 82   2                 DCL KEY ADDRESS,
                           ( LOCK BASED KEY) BYTE;


 83   2         SETL1:      DO WHILE LOCK <> 0;
 84   3                   END;
 85   2                  LOCK = CPTR;
                           /* SET LOCK */
 86   2                 IF  LOCK <> CPTR THEN GOTO SETL1;
                           /* ALREADY LOCKED BY OTHER COMPUTER */
 88   2                 RETURN;
 89   2                 END GET;
```

```
                $EJECT
                /*
                ***********************************************************
                ***********************************************************
                **

                                                                    RELEASE
                    RELEASES A VARIABLE LOCK
                    INPUT: VARIABLE POINTER.

                **
                ***********************************************************
                */
90    1         RELEASE:   PROCEDURE(KEY) PUBLIC;
91    2                       DCL KEY ADDRESS,
                                  (LOCK BASED KEY) BYTE;

92    2                     LOCK = 0;
93    2                    RETURN;
94    2                    END RELEASE;
```

```
                $EJECT
                /*
                ***********************************************************
                ***********************************************************
                **
                                                              SETEXTMSG

                    COMPUTE NUMBER OF RECEIVING COMPUTER AND SET
                    VARIABLE EXTMSG
                **
                ***********************************************************
                */
 95    1        SETEXTMSG:  PROCEDURE PUBLIC;

 96    2        DCL      (X1,X2,X3) BYTE;

 97    2                 IF NUMEXTMSG = 0
                             THEN EXTMSG = 0;
 99    2                   ELSE DO;
100    3                     X1 = 8;
101    3                     X2 = EXTMSGBUFFER(EXTMSGOUT);
                                 /* RMN */
102    3                     DO X3 = 1 TO MAXCPTR;
103    4                       IF X2 < X1 THEN GOTO SET1;
105    4                       X1 = X1 + 8;
106    4                       END;
107    3        SET1:            EXTMSG = X3;
108    3                     END;
109    2                 RETURN;
110    2                 END SETEXTMSG;
```

```
                $EJECT
                /*
                ***********************************************************
                ***********************************************************
                                                                    RECEXT
                    TRANSFER MSG FROM EXTERNAL MSG BUFFER INTO LOCAL
                    MSG BUFFER
                **
                ***********************************************************
                */
111    1        RECEXT:     PROCEDURE PUBLIC;
112    2        DCL      (X1,X2,X3) BYTE;

113    2                  CALL GET(.EXTMSGLOCK);
                               /* SET EXTMSGBUFFER LOCK */
114    2                  IF EXTMSGOUT = EXTLASTMSG THEN EXTLASTMSG,EXTMSGOUT = 0;
                               /* NEXT MSG AT TOP OF BUFFER */
116    2                  X1 = EXTMSGOUT;
117    2                  X2 = EXTMSGBUFFER(X1 + ML);
                               /* LENGTH OF MSG */
118    2                  EXTMSGOUT = X1 + X2;
                               /* COMPUTE BEGIN OF NEXT MSG */
119    2                  IF NOT SEND (.EXTMSGBUFFER(X1)) THEN GOTO REC1;
                               /* OVERFLOW */
121    2                  IF X2 = 4 THEN GOTO REC1;
                               /* NO DATA BYTES */
123    2                  X1 = X1 + 4;
124    2                  X2 = X2 - 4;
125    2                      CALL MOVE(X2,.EXTMSGBUFFER(X1),ADRMSGDATA);
                               /* TRANSFER DATA BYTES */
126    2        REC1:         NUMEXTMSG = NUMEXTMSG - 1;
127    2                  CALL SETEXTMSG;
                               /* SET NUMBER OF RECEIVING CPTR OF NEXT MSG */
128    2                  CALL RELEASE(.EXTMSGLOCK);
129    2                  RETURN;
130    2                  END RECEXT;
```

```
                $EJECT
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                        SENDEXT

                    SEND MSG TO EXTERNAL MODULE
                    ADRMSG POINTS TO FIRST BYTE OF MSG
                    OUTPUT: TRUE IF MSG SENT
                            FALSE OTHERWISE
                **
                **********************************************************************
                */
  131   1       SENDEXT:    PROCEDURE PUBLIC;

  132   2                 CALL GET(.EXTMSGLOCK);
                              /* SET EXT MSG BUFFER LOCK */
  133   2                 IF EXTMSGIN = EXTMSGOUT AND NUMEXTMSG <> 0 THEN GOTO SEXT2;
                              /* OVERFLOW */
  135   2                 IF EXTMSGOUT <= EXTMSGIN THEN
  136   2                   DO;
  137   3                   IF LAST(EXTMSGBUFFER) - EXTMSGIN >= MSG(ML) THEN GOTO SEXT1;
                                /* MSG FITS INTO REST OF BUFFER */
  139   3                   EXTLASTMSG = EXTMSGIN;
  140   3                   EXTMSGIN = 0;
  141   3                   END;
  142   2                 IF EXTMSGOUT - EXTMSGIN < MSG(ML) THEN GOTO SEXT2;
                              /* OVERFLOW */

  144   2       SEXT1:        XB2 = MSG(ML) ;
  145   2                         CALL MOVE(XB2,ADR$MSG,.EXTMSGBUFFER(EXTMSGIN));
  146   2                         EXTMSGIN = EXTMSGIN + XB2 ;
                              /* TRANSFER MSG */
  147   2                 IF EXTMSGIN > LAST(EXTMSGBUFFER) THEN
                              /* EXTMSGIN OUTSIDE BUFFER */
```

```
148   2                 DO;
149   3                 EXTLASTMSG = EXTMSGIN;
150   3                 EXTMSGIN = 0;
151   3                 END;
152   2               NUMEXTMSG = NUMEXTMSG + 1;
153   2               IF NUMEXTMSG = 1 THEN CALL SETEXTMSG;
                          /* SET RECEIVING CPTR OF NEXT MSG */
155   2               CALL RELEASE(.EXTMSGLOCK);
                          /* RELEASE(.EXTMSGLOCK) EXT MSG BUFFER */
156   2               RETURN;
157   2       SEXT2:      CALL MSGOVERFLOW;
158   2               CALL RELEASE(.EXTMSGLOCK);
159   2               RETURN;
160   2               END SENDEXT;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                ILLEGALMSG

                    PROCESS ILLEGAL MSG RECEIVED BY A MODULE
                **
                ***************************************************************
                */
 161    1       ILLEGALMSG: PROCEDURE PUBLIC;

 162    2       DCL         ILLEGAL(13) BYTE DATA('ILLEGAL MSG. '),
                            ILLMSG(25) BYTE;

 163    2                 XB2 = 0;
 164    2                 DO XB1 = 0 TO LAST(MSG);
 165    3                   CALL CONVASC(MSG(XB1));
 166    3                   ILLMSG(XB2) = BU;
 167    3                   ILLMSG(XB2 + 1) = BL;
 168    3                       ILLMSG(XB2 + 2) = ' ';
 169    3                   XB2 = XB2 + 3;
 170    3                   END;
                              /* CONVERT RMN,SMN,MN,ML */
 171    2                 CALL MOVE(13,.ILLEGAL,.ILLMSG(12));
 172    2                 CALL PRINT$ASYNC(.ILLMSG,25);
 173    2                 RETURN;
 174    2                 END ILLEGALMSG;
 175    1       END SCPUB3;
```

196

MODULE INFORMATION:

CODE AREA SIZE = 03A3H 931D

```
     VARIABLE AREA SIZE = 0029H      41D
     MAXIMUM STACK SIZE = 0008H       8D
     448 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB4
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SC4.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            SCPUB4:     DO;
                $NOLIST

  26    1       LEGAL: PROCEDURE (P1,P2,P3) BYTE EXTERNAL;
  27    2              DCL (P1,P2,P3) BYTE;
  28    2              END;
  29    1       SET$BIT: PROCEDURE (P1,P2) EXTERNAL;
  30    2                 DCL P1 BYTE, P2 ADDRESS;
  31    2                 END;
  32    1       SHFT: PROCEDURE(P1) BYTE EXTERNAL;
  33    2             DCL P1 BYTE;
  34    2             END;
  35    1       CLEAR$BIT: PROCEDURE (P1,P2) EXTERNAL;
  36    2                    DCL P1 BYTE, P2 ADDRESS;
  37    2                    END;
  38    1       ERROR: PROCEDURE (P1,P2,P3) EXTERNAL;
  39    2              DCL P1 ADDRESS, (P2,P3) BYTE;
  40    2              END;
```

```
                $EJECT
                /*
                ***********************************************************
                ***********************************************************
                **
                                                                PRIORITY

                    SCHEDULE THE CALL OF A PRIORITY PROCEDURE
                    (PRIORITYINT TO BE CALLED FROM INT ROUTINES ONLY)
                        INPUT: PRIORITY INDEX
                **
                ***********************************************************
                */
 41    1        PRIORITY: PROCEDURE (P1) PUBLIC;

 42    2        DCL     P1 BYTE;

 43    2                IF P1 >= MAXPRIOR THEN RETURN;
                             /* ILLEGAL PRIORITY */
 45    2                CALL SET$BIT(P1,.PRIORSCHEDULE);
 46    2                RETURN;
 47    2                END PRIORITY;
```

```
                $EJECT
                /*
                ***********************************************************************
                ***********************************************************************
                **
                                                                        ENTERPRIOR

                    ENTER NEW PRIORITY CALL IN PRIORITY LIST
                        INPUT  : ADDRESS OF PRIORITY PROCEDURE
                        OUTPUT : PRIORITY INDEX TO PRIORLIST
                **
                ***********************************************************************
                */
 48   1         ENTERPRIOR: PROCEDURE (P1) BYTE PUBLIC;

 49   2         DCL     P1 ADDRESS;

 50   2                 DO XB1 = 0 TO LAST(PRIORLIST);
                       /* SEARCH FOR FREE SLOT IN PRIORLIST */
 51   3                   IF PRIORLIST(XB1) = 0 THEN
 52   3                     DO;
 53   4                     PRIORLIST(XB1) = P1;
                             /* ADDRESS OF PRIORITY PROCEDURE */
 54   4                     RETURN XB1;
 55   4                     END;
 56   3                   END;
 57   2                 CALL ERROR(.ENTERPRIOR,0,2);
                       /* PRIORITY LIST OVERFLOW */
 58   2                 RETURN 0FFH;
 59   2                 END ENTERPRIOR;
```

```
                $EJECT
                /*
                ***********************************************************
                ***********************************************************
                **
                                                                REMPRIOR

                    REMOVE PRIORITY CALL FROM PRIORLIST
                        INPUT: PRIORITY INDEX
                **
                ***********************************************************
                */
60    1         REMPRIOR:   PROCEDURE (P1) PUBLIC;

61    2         DCL     P1 BYTE;

62    2                 IF NOT LEGAL(P1,MAXPRIOR,6) THEN RETURN;
                            /* ATTEMPT TO REMOVE PRIORITY WITH ILLEGAL INDEX */
64    2                 CALL CLEARBIT(P1,.PRIORSCHEDULE);
                            /* CLEAR SCHEDULE BIT IF SET */
65    2                 PRIORLIST(P1) = 0;
                            /* REMOVE PRIORITY ADDRESS */
66    2                 RETURN;
67    2                 END REMPRIOR;
```

```
                $EJECT
                /*
                ***********************************************************
                ***********************************************************
                **
                                                                SETCDC

                        SET CDC INTERRUPT
                        INPUT : P1 - TIME INTERVAL (LSB = 1.86 MICRO SEC
                        P2 - ADDRESS TO BE CALLED AT CDC INT
                        OUTPUT: FALSE IF CDC ALREADY ACTIVE
                                TRUE OTHERWISE
                **
                ***********************************************************
                */
 68   1         SETCDC:     PROCEDURE (P1,P2) BYTE PUBLIC;

 69   2         DCL         (P1,P2) ADDRESS,
                          (P1H,P1L) BYTE AT (.P1);

 70   2                   IF CDCACTIVE THEN RETURN FALSE;
 72   2                   CDCACTIVE = TRUE;
 73   2                   DISABLE;
 74   2                   INTMASK = INTMASK XOR 1 ;
 75   2                   OUTPUT(0DBH) = INTMASK;
 76   2                   OUTPUT(0DDH) = P1H;
 77   2                   OUTPUT(0DDH) = P1L;
                              /* LOAD CTR 1 */
 78   2                   CDCADR = P2;
                              /* SAVE ADRRESS TO BE CALLED AT CDC INT */
 79   2                   ENABLE;
 80   2                   RETURN TRUE;
 81   2                   END SETCDC;
```

202

```
                $EJECT
                /*
                ****************************************************************
                ****************************************************************
                **
                                                                ENTER$INT
                ENTER INTERRUPT:P1-PRIORITY LEVEL
                **
                ****************************************************************
                */
  82    1       ENTER$INT: PROCEDURE(P1) PUBLIC;
  83    2                  DCL P1 BYTE;

  84    2                  INTMASK = INTMASK XOR SHFT(P1);
  85    2                  DISABLE;
  86    2                  OUTPUT(0DBH) = INTMASK;
  87    2                  ENABLE;
  88    2                  RETURN;
  89    2                  END;
```

```
                $EJECT
                /*
                **********************************************************
                **********************************************************

                                                                REM$INT
                REMOVE INTERRUPT: P1 INTERRUPT LEVEL

                **********************************************************
                */
  90    1       REM$INT: PROCEDURE(P1) PUBLIC;
  91    2                DCL P1 BYTE;

  92    2                INTMASK = INTMASK OR SHFT(P1);
  93    2                DISABLE;
  94    2                OUTPUT(0DBH) = INTMASK;
  95    2                ENABLE;
  96    2                RETURN;
  97    2                END;
  98    1       END SCPUB4;
```


MODULE INFORMATION:

```
     CODE AREA SIZE     = 0102H    258D
     VARIABLE AREA SIZE = 000AH     10D
     MAXIMUM STACK SIZE = 0004H      4D
     308 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB5
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SC5.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            SCPUB5:     DO;
                $NOLIST

  26    1       SEND: PROCEDURE (P1) BYTE EXTERNAL;
  27    2             DCL P1 ADDRESS;
  28    2             END;
                /*
                ***************************************************************
                ***************************************************************
                **
                                                              PRINT$ASYNC
                PRINT MESSAGE ON CONSOLE
                INPUT: P1 - ADDRESS OF TEXT
                       P2 - NUMBER OF BYTES ON TEXT

                ***************************************************************
                */
  29    1       PRINT$ASYNC: PROCEDURE(P1,P2) PUBLIC;
  30    2                    DCL P1 ADDRESS, P2 BYTE;

  31    2                    SYSMN10(1) = EX; /* EXECUTIVE ASSUME RESPONSABILITY */
  32    2                    SYSMN10(3) = P2 + 4; /* MSG LENGHT */

  33    2                    IF NOT SEND(.SYSMN10) THEN RETURN;
  35    2                    CALL MOVE(P2,P1,ADRMSGDATA); /* SEND MESSAGE */

  36    2                    RETURN;
  37    2                    END PRINT$ASYNC;
```

```
                $EJECT
                /*
                ****************************************************************
                ****************************************************************
                **
                                                                    PRINT$SYNC

                PRINT MESSAGE ON CONSOLE
                INPUT: P1 - ADDRESS OF TEXT
                       P2 - NUMBER OF BYTES ON TEXT
                       P3 - SENDER MODULE NUMBER

                ****************************************************************
                */
  38    1       PRINT$SYNC: PROCEDURE(P1,P2,P3) PUBLIC;
  39    2                   DCL P1 ADDRESS,(P2,P3) BYTE;

  40    2                   SYSMN11(1) = P3;
  41    2                   SYSMN11(3) = P2 + 4; /* MSG LENGHT */

  42    2                   IF NOT SEND(.SYSMN11) THEN RETURN;
  44    2                   CALL MOVE(P2,P1,ADRMSGDATA);

  45    2                   RETURN;
  46    2                   END PRINT$SYNC;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                CON$INPUT$REQ

                REQUEST INPUT FROM CONSOLE
                INPUT: P1 - SENDER MODULE NUMBER
                       P2 - ROLL SCREEN FLAG
                            (IF TRUE ROLL ECREEN AFTER INPUT).

                ***************************************************************
                */
47    1         CON$INPUT$REQ: PROCEDURE(P1,P2) PUBLIC;
48    2                        DCL (P1,P2) BYTE;

49    2                        SYSMN12(1) = P1 ;/* SENDER NUMBER */
50    2                        IF NOT SEND(.SYSMN12) THEN RETURN;
52    2                        MSGDATA(0) = P2;

53    2                        RETURN;
54    2                        END CON$INPUT$REQ;
```

```
                $EJECT
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                    CON$LINK$REQ

                REQUEST LINKAGE WITH CONSOLE
                INPUT: P1 - SENDER MODULE NUMBER

                **********************************************************************
                */
 55    1        CON$LINK$REQ: PROCEDURE(P1) PUBLIC;
 56    2                      DCL P1 BYTE;

 57    2                      SYSMN13(1) = P1; /* SENDER NUMBER */
 58    2                      B1 = SEND(.SYSMN13);

 59    2                      RETURN;
 60    2                      END CON$LINK$REQ;
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                    PAR$LINK$REQ

                REQUEST LINKAGE WITH PRINTER
                INPUT: P1 - SENDER MODULE NUMBER
                **********************************************************************
                */
 61    1        PAR$LINK$REQ: PROCEDURE(P1) PUBLIC;
 62    2                      DCL P1 BYTE;

 63    2                      SYSMN14(1) = P1 ;/* SENDER NUMBER */
 64    2                      B1 = SEND(.SYSMN14);

 65    2                      RETURN;
 66    2                      END;
```

```
                $EJECT
                /*
                ****************************************************************
                ****************************************************************
                **
                                                              CON$RELEASE
                RELEASE LINKAGE WITH CONSOLE
                :
                ****************************************************************
                */
67    1         CON$RELEASE: PROCEDURE PUBLIC;

68    2                      B1 = SEND(.SYSMN15);
69    2                      RETURN;
70    2                      END;
                /*
                ****************************************************************
                ****************************************************************
                **
                                                              CON$NEW$LINE
                BEGIN NEW LINE ON CONSOL
                INPUT: P1 - SENDER MODULE NUMBER

                ****************************************************************
                */
71    1         CON$NEW$LINE: PROCEDURE(P1) PUBLIC;
72    2                       DCL P1 BYTE;

73    2                       SYSMN16(1) = P1;
74    2                       B1 = SEND(.SYSMN16);
75    2                       RETURN;
76    2                       END CON$NEW$LINE;
```

```
$EJECT
/*
**********************************************************************
**********************************************************************
**
                                                        CON$BEG$LINE
BEGIN NEW TEXT ON CONSOLE
INPUT: P1 - SENDER NUMBER

**********************************************************************
*/
CON$BEG$LINE: PROCEDURE(P1) PUBLIC;
                  DCL P1 BYTE;

                  SYSMN17(1) = P1;
                  IF NOT SEND(.SYSMN17) THEN RETURN;
                  RETURN;
                  END CON$BEG$LINE;
```

```
                $EJECT
                /*
                ***********************************************************************
                ***********************************************************************
                :*
                                                                        PRINT$LINKED
                PRINT ON CONSOLE AND ASK FOR ACKNOWLEDGE
                INPUT: P1 - ADDRESS OF TEXT
                       P2 - NUMBER OF BYTE ON TEXT
                       P3 - SENDER MODULE NUMBER

                ***********************************************************************
                */
  84    1       PRINT$LINKED: PROCEDURE(P1,P2,P3,P4) PUBLIC;
  85    2                     DCL P1 ADDRESS,(P2,P3,P4) BYTE;

  86    2                     SYSMN18(1) = P3;
  87    2                     SYSMN18(3) = P2 + 5;

  88    2                     IF NOT SEND(.SYSMN18) THEN RETURN;
  90    2                     MSGDATA(0) = P4; /* TELLBACK CODE */
  91    2                     CALL MOVE(P2,P1,ADRMSGDATA+1);

  92    2                     RETURN;
  93    2                     END PRINT$LINKED;
```

```
                $EJECT
                /*
                **********************************************************
                **********************************************************
                **
                                                        PAR$RELEASE
                RELEASE LINKAGE WITH PRINTER
                :
                **********************************************************
                */
 94    1        PAR$RELEASE: PROCEDURE PUBLIC;

 95    2                     B1 = SEND(.SYSMN19);
 96    2                     RETURN;
 97    2                     END;
                /*
                **********************************************************
                **********************************************************
                **
                                                            PAGE
                REQUEST A NEW PAGE IN SYSTEM PRINTER
                **
                **********************************************************
                */
 98    1        PAGE: PROCEDURE (P1) PUBLIC;
 99    2              DCL P1 BYTE;

100    2              SYSMN24(1) = P1;
101    2              IF NOT SEND(.SYSMN24) THEN RETURN;
103    2              RETURN;
104    2              END;
```

```
                $EJECT
                /*
                **********************************************************
                **********************************************************
                **
                                                                    WRITE
                WRITE IN SYSTEM PRINTER
                **
                **********************************************************
                */
105    1        WRITE: PROCEDURE (P1,P2,P3) PUBLIC;
106    2               DCL P1 ADDRESS, (P2,P3) BYTE;

107    2               SYSMN23(1) = P3; SYSMN23(3) = P2 + 4 ;
109    2               IF NOT SEND(.SYSMN23) THEN RETURN;
111    2               CALL MOVE(P2,P1,ADRMSGDATA);
112    2               RETURN;
113    2               END;
```

```
                $EJECT
                /*
                ***********************************************************
                ***********************************************************
                **
                                                          WRITE$LINKED
                WRITE IN SYSTEM PRINTER
                AND WAIT FOR AN ANSWER
                **
                ***********************************************************
                */
 114    1       WRITE$LINKED: PROCEDURE (P1,P2,P3,P4) PUBLIC;
 115    2                     DCL P1 ADDRESS, (P2,P3,P4) BYTE;

 116    2                     SYSMN25(1)=P3; SYSMN25(3)=P2 + 5 ;
 118    2                     IF NOT SEND(.SYSMN25) THEN RETURN;
 120    2                     MSGDATA(0) = P4 ; /* TELLBACK CODE */
 121    2                     CALL MOVE(P2,P1,ADRMSGDATA+1);
 122    2                     RETURN;
 123    2                     END;

 124    1       END SCPUB5;
```

214

MODULE INFORMATION:

```
     CODE AREA SIZE     = 01E3H    483D
     VARIABLE AREA SIZE = 001CH     28D
     MAXIMUM STACK SIZE = 0004H      4D
     397 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB6
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SC6.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            SCPUB6:     DO;
                $NOLIST
  26    1       BIT: PROCEDURE (P1,P2) BYTE EXTERNAL;
  27    2            DCL (P1,P2) BYTE;
  28    2            END;
               /*
               ***********************************************************************
               ***********************************************************************
               **
                                                                          ACTIVE
               CHECK ACTIVE STATUS OF MODULE
               INPUT: MODULE NUMBER
               OUTPUT: TRUE IF MODULE ACTIVATED,
                       FALSE OTHERWISE.
               **
               ***********************************************************************
               */
  29    1       ACTIVE: PROCEDURE (P1) BYTE PUBLIC;
  30    2               DCL P1 BYTE;

  31    2               RETURN BIT(1,MODSTATUS(P1));

  32    2               END ACTIVE;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                    UPDSTAT

                    UPDATE MODULE STATUS IN MODSTATUS TABLE
                        INPUT: P1 - MODULE NUMBER
                               P2 - STATUS
                **
                ***************************************************************
                */
  33    1       UPDSTAT:    PROCEDURE (P1,P2) PUBLIC;

  34    2       DCL         (P1,P2) BYTE;

  35    2                   MODSTATUS(P1) = P2;
  36    2                   RETURN;
  37    2                   END UPDSTAT;
```

```
                $EJECT
                /*
                ***********************************************************
                ***********************************************************
                **
                                                                    CONVASC

                    CONVERT 2 DIGIT HEX NUMBER INTO 2 ASCII CODES
                        INPUT  : NUMBER (BYTE)
                        OUTPUT : ASCII CODES IN A4
                **
                ***********************************************************
                */
  38    1       CONVASC:    PROCEDURE (P1) PUBLIC;

  39    2       DCL         P1 BYTE;

  40    2                   BL = (P1 AND 00001111B) + 30H;
  41    2                   IF BL > 39H THEN BL = BL + 7;
  43    2                   BU = ROR((P1 AND 11110000B),4) + 30H;
  44    2                   IF BU > 39H THEN BU = BU + 7;
  46    2                   RETURN;
  47    2                   END CONVASC;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************
                **
                                                                        DEBLK

                    MOVE POINTER B1 TO 1ST NOM-BLANK IN MSGDATA
                    STARTING AT MSGDATA(B1)
                **
                ***************************************************************
                */
 48    1        DEBLK:      PROCEDURE PUBLIC;

 49    2                 DO WHILE MSGDATA(B1) = ' ' AND B1 <= MSG(ML) - 5;
 50    3                   B1 = B1 + 1;
 51    3                   END;
 52    2                 RETURN;
 53    2                 END DEBLK;
```

```
                $EJECT
                /*
                ****************************************************************
                ****************************************************************
                **
                                                                    GETNUM

                    CONVERT ASCII CODES IN MSGDATA TO HEX NUMBER
                    STARTIN AT MSGDATA(B1)
                    CONVERTED NUMBER IS LEFT IN A4
                    RETURN TRUE IF NUMBER IS O.K.,FALSE OTHERWISE
                **
                ****************************************************************
                */
 54   1         GETNUM: PROCEDURE BYTE PUBLIC;
 55   2             DCL EOT LIT '04H'; /* END OF TASK ID */
 56   2                 DCL (BASE10,BASE16) ADDRESS,
                         MORE$STRING BYTE,
                         DIGITS(*) BYTE DATA('0123456789ABCDEF');

 57   2             CALL DEBLK; /* SKEEP BLANKS */
 58   2             IF MSGDATA(B1) = EOT THEN RETURN FALSE;

 60   2                 BASE10,BASE16 = 0 ; MORE$STRING = TRUE ;
 62   2                 DO WHILE MORE$STRING;
 63   3                    MORE$STRING = FALSE ;
 64   3                    XB1 = MSGDATA(B1) ;
 65   3                    DO XB2 = 0 TO LAST(DIGITS);
 66   4                       IF XB1 = DIGITS(XB2) THEN
 67   4                          DO;
 68   5                          BASE10 = SHL(BASE10,3) + SHL(BASE10,1) + XB2 ;
 69   5                          BASE16 = SHL(BASE16,4) + XB2 ;
 70   5                          B1 = B1 + 1 ;
 71   5                          MORE$STRING = TRUE ;
 72   5                          END;
 73   4                    END;
```

```
74   3              END;
75   2           IF XB1 = 'H' THEN DO;
77   3                           A4 = BASE16 ; B1 = B1 + 1 ;
79   3                           RETURN TRUE;
80   3                           END;
81   2           IF XB1 = ' ' OR XB1 = '-' OR XB1 = ',' OR
                    XB1 = '.' THEN DO;
83   3                           A4 = BASE10 ; B1 = B1 + 1 ;
85   3                           RETURN TRUE;
86   3                           END;
87   2           IF XB1 = EOT THEN DO;
89   3                           A4 = BASE10;
90   3                           RETURN TRUE;
91   3                           END;
92   2           RETURN FALSE;
93   2           END;
```

```
                $EJECT
                /*
                ***********************************************************
                ***********************************************************
                **
                                                          VECTOR$ADD
                ADD TWO 4 BYTES VECTORS
                **
                ***********************************************************
                */
 94    1        VECTOR$ADD: PROCEDURE (P1,P2,P3) PUBLIC;
 95    2                    DCL (P1,P2,P3) ADDRESS,
                               (ELE1 BASED P1) BYTE,
                               (ELE2 BASED P2) BYTE,
                               (ELE3 BASED P3) BYTE,
                               CY BYTE;

 96    2                    P1 = P1 + 3;
 97    2                    P2 = P2 + 3;
 98    2                    P3 = P3 + 3;
 99    2                    CY = 0 ;
100    2                    DO XB1 = 0 TO 3;
101    3                       ELE3 = ELE1 + ELE2 + CY;
102    3                       CY = 0 PLUS 0 ;
103    3                       P1=P1-1; P2=P2-1; P3=P3-1;
106    3                       END;
107    2                    RETURN;
108    2                    END;
```

221

```
                $EJECT
                /*
                ****************************************************************
                ****************************************************************
                **
                                                              VECTOR$SUB
                SUBSTRACT TWO 4 BYTE VECTORS
                **
                ****************************************************************
                */
  109   1       VECTOR$SUB: PROCEDURE (P1,P2,P3) PUBLIC;
  110   2                   DCL (P1,P2,P3) ADDRESS,
                               (ELE1 BASED P1) BYTE,
                               (ELE2 BASED P2) BYTE,
                               (ELE3 BASED P3) BYTE,
                               CY BYTE;

  111   2                   P1 = P1 + 3;
  112   2                   P2 = P2 + 3;
  113   2                   P3 = P3 + 3;
  114   2                   CY = 1;
  115   2                   DO XB1 = 0 TO 3;
  116   3                      ELE3 = ELE1 - ELE2 + CY - 1 ;
  117   3                      CY = 2 MINUS 1 ;
  118   3                      P1=P1-1; P2=P2-1; P3=P3-1;
  121   3                      END;
  122   2                   RETURN;
  123   2                   END;
```

```
                $EJECT
                /*
                **********************************************************
                **********************************************************
                **
                                                              NUMOUT
                PREPARE STRING FOR OUTPUT
                **
                **********************************************************
                */
  124    1      NUMOUT: PROCEDURE(VALUE,BASE,LC,BUF$ADR,WIDTH) PUBLIC;
  125    2      DCL (VALUE,BUF$ADR) ADDRESS,
                    (BASE,LC,WIDTH) BYTE,
                    (CHAR BASED BUF$ADR) (1) BYTE,
                    DIGITS(*) BYTE DATA('0123456789ABCDEF');

  126    2          DO XB1 = 1 TO WIDTH;
  127    3             CHAR(WIDTH-XB1) = DIGITS(VALUE MOD BASE);
  128    3             VALUE = VALUE / BASE;
  129    3             END;
  130    2          XB1 = 0;
  131    2          DO WHILE CHAR(XB1) = '0' AND XB1 < WIDTH - 1 ;;
  133    3             CHAR(XB1) = LC ;
  134    3             XB1 = XB1 + 1 ;
  135    3             END;
  136    2          RETURN;
  137    2          END NUMOUT;
```

```
                $EJECT
                /*
                ***********************************************************
                ***********************************************************
                **
                                                              GET$CHAR
                GET A CHARACTER FROM INPUT STRING
                *
                ***********************************************************
                */
 138    1       GET$CHAR: PROCEDURE BYTE PUBLIC;
 139    2           DCL EOT LIT '04H'; /* END OF TEXT ID */

 140    2           CALL DEBLK; /* SKEEP BLANKS */
 141    2           IF MSGDATA(B1) = EOT THEN RETURN FALSE;

 143    2           BL = MSGDATA(B1); B1 = B1 + 1 ;
 145    2           RETURN TRUE;
 146    2           END GET$CHAR;
 147    1       END SCPUB6;
```



```
MODULE INFORMATION:

     CODE AREA SIZE     = 0366H    870D
     VARIABLE AREA SIZE = 001EH     30D
     MAXIMUM STACK SIZE = 0006H      6D
     381 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE SCPUB7
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:SC7.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            SCPUB7:   DO;
                $NOLIST
  26    1       NUMOUT: PROCEDURE (P1,P2,P3,P4,P5) EXTERNAL;
  27    2               DCL (P1,P4) ADDRESS, (P2,P3,P5) BYTE;
  28    2               END;
  29    1       PRINT$ASYNC: PROCEDURE (P1,P2) EXTERNAL;
  30    2                    DCL P1 ADDRESS, P2 BYTE;
  31    2                    END;
  32    1       CON$LINK$REQ: PROCEDURE (P1) EXTERNAL;
  33    2                     DCL P1 BYTE;
  34    2                     END;
  35    1       UPDSTAT: PROCEDURE (P1,P2) EXTERNAL;
  36    2                DCL (P1,P2) BYTE;
  37    2                END;
```

```
                $EJECT
                /*
                ****************************************************************
                ****************************************************************
                **
                                                        ENTER$MSG$MOD

                ENTER MSG ADDRESS INTO MESSAGE ADDRESS TABLE
                    INPUT: P1 - MESSAGE MODULE NUMBER.
                           ADR - MESSAGE MODULE ADDRESS.

                ****************************************************************
                */
  38    1       ENTER$MSG$MOD: PROCEDURE (P1,ADR) PUBLIC;

  39    2           DCL P1 BYTE, ADR ADDRESS;

  40    2           CALL UPDSTAT(P1,1); /* UPDATE MODULE EXIST */
  41    2           MSG$MOD$ADDRESS(P1) = ADR;/* SET MSG ADDRESS */
  42    2           RETURN;
  43    2           END;

                /*
                ****************************************************************
                ****************************************************************
                **
                                                            MONITOR

                ****************************************************************
                */
  44    1       MONITOR: PROCEDURE PUBLIC;
  45    2           DCL MON$TAG (18) BYTE DATA('MONITOR COMPUTER ',CPTR+30H);

  46    2           CALL PRINT$ASYNC(.MON$TAG,18);
  47    2           CALL CON$LINK$REQ(EX);
  48    2           RETURN;
```

```
                $EJECT
                /*
                **********************************************************************
                **********************************************************************
                **
                                                                        ERROR

                    SYSTEM ERROR
                    MAY BE CALLED WHEN AN INTERNAL ERROR CONDITION OCCURS,
                    PROGRAM LIMITS ARE EXCEEDED ETC.
                    ERROR GENERATES AN ASYNCHRONOUS PRINTOUT TO INDICATE
                    THE NATURE AND LOCATION OF ERROR CONDITION .
                        INPUT: P1 - LOCATION OF ERROR OR OTHER ADDRESS VALUE
                               P2,P3 - BYTE VALUES TO SPECIFY ERROR
                **
                **********************************************************************
                */
 50   1         ERROR :     PROCEDURE (P1,P2,P3) PUBLIC;

 51   2         DECLARE     P1 ADDRESS,(P2,P3) BYTE,
                         (STACKTOP BASED P1) ADDRESS;

 52   2             CALL NUMOUT(STACKTOP,16,' ',.ERRORMSG(7),4);
 53   2             CALL NUMOUT(DOUBLE(P2),10,' ',.ERRORMSG(11),3);
 54   2             CALL NUMOUT(DOUBLE(P3),10,' ',.ERRORMSG(14),3);
 55   2             CALL PRINT$ASYNC(.ERRORMSG,17);
                            /* SEND ASYNC PRINT MSG TO CRT */
 56   2             RETURN;
 57   2             END ERROR;
 58   1         END SCPUB7;
```

227

MODULE INFORMATION:

CODE AREA SIZE = 0098H 152D

```
     VARIABLE AREA SIZE = 0007H      7D
     MAXIMUM STACK SIZE = 0008H      8D
     223 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

EMULATOR PROGRAM LISTING

The following table summarizes the contents of emulator files.

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE EMULA1
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:EMULA1.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            EMULA1:
                /**************************************************************
                **************************************************************

                THIS MODULE CONTAINS THE PERIODIC AND DEMAND EMULATOR PROCEDURES
                ,THE NETWORK INFORMATION DATA RECEIVER AND THE ROUTINE TO INITIA-
                TE THEM INTO THE SYSTEM.
                **************************************************************
                */
                DO;
                $NOLIST

231
  99   1        DCL DBD LIT 'DATA$BLOCK$DEM',
                    DBP LIT 'DATA$BLOCK$PER',
                    DTA LIT 'MSG$DATA',
                    MBK LIT 'MOD$BLOCK';
```

```
                $EJECT
                /*
                **********************************************************

                SEND MESSAGES FROM SOURSE TO SINK TASK
                INPUT: POINTER TO ELEMENT NUM$MSG$OUT IN ACTIVATION RECORD.
                **********************************************************
                */
100    1        SEND$MSGS: PROCEDURE (ADR$ELEMENT);
101    2            DCL NUM BYTE, ADR$ELEMENT ADDRESS;
102    2            DCL (MSG$ELEMENT BASED ADR$ELEMENT) BYTE;

103    2            MSG$HEADER (SMN) = CPTR$ID + 1 ;
104    2            NUM = MSG$ELEMENT;
105    2            IF NUM = 0 THEN RETURN;

107    2            CALL COPY$CLOCK(.MSG$TEXT(1)); /* GET SENDING TIME */

108    2            DO B1 = 1 TO NUM;        /* SEND ALL MSGS   */
109    3                ADR$ELEMENT = ADR$ELEMENT + 1 ;
110    3               MSG$HEADER (RMN) = MSG$ELEMENT;
111    3                ADR$ELEMENT = ADR$ELEMENT + 1 ;
112    3               MSG$HEADER (MN)  = MSG$ELEMENT;
113    3                ADR$ELEMENT = ADR$ELEMENT + 1 ;
114    3               MSG$TEXT (0)     = MSG$ELEMENT;
115    3                ADR$ELEMENT = ADR$ELEMENT + 1 ;
116    3               MSG$HEADER (ML)  = MSG$ELEMENT + 9 ;
117    3               IF NOT SEND(.MSG$HEADER) THEN CALL ILLEGALMSG;
119    3                  ELSE CALL MOVE(5,.MSG$TEXT,ADRMSGDATA);
120    3               END;
121    2            RETURN;
122    2            END SEND$MSGS;
```

232

```
                $EJECT
                /*
                **********************************************************************
                **********************************************************************

                EMULATE: EMULATE DELAY TIME AND GET CONSUMED TIME
                INPUT: POINTER TO ELEMENT DELAY IN ACTIVATION RECORD.
                **********************************************************************
                */
 123   1        EMULATE: PROCEDURE (DAT$ADR);
 124   2            DCL DAT$ADR ADDRESS;
 125   2            DCL (DAT BASED DAT$ADR) BYTE;

 126   2            DO B1 = 1 TO DAT; /* NUMBER OF MILLISECONDS */
 127   3               CALL TIME (10);
 128   3               END;

 129   2                DAT$ADR = DAT$ADR + 1 ;
 130   2            DAT = DAT + 1 ; /* INCREMENT NUMBER OF EXECUTIONS */
 131   2                DAT$ADR = DAT$ADR + 1;
 132   2            CALL COPY$CLOCK(.TEMP$CLOCK2);  /* GET ACTUAL TIME */
 133   2            CALL VECTOR$SUB(.TEMP$CLOCK2,.TEMP$CLOCK1,.TEMP$CLOCK3);
 134   2            CALL VECTOR$ADD(.TEMP$CLOCK3,DAT$ADR,DAT$ADR);
 135   2            RETURN;
 136   2            END EMULATE;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************

                CORRECT$INPUT$MSG: VERIFIES WHETHER THE MSG JUST ARRIVED IS O.K.
                INPUT: MSG$ID - ARRIVED MESSAGE NUMBER.
                       ADR$ELEMENT - POINTER TO ELEMENT NUM$MSG$IN
                                     IN ACTIVATION RECORD.
                ***************************************************************
                */
  137   1       CORRECT$INPUT$MSG: PROCEDURE (MSG$ID,ADR$ELEMENT) BYTE;
  138   2           DCL (MSG$ID,NUM$MSG) BYTE, ADR$ELEMENT ADDRESS;
  139   2           DCL (ELEMENT BASED ADR$ELEMENT) BYTE;

  140   2           NUM$MSG = ELEMENT;
  141   2           IF NUM$MSG = 0 THEN RETURN FALSE;
  143   2           ADR$ELEMENT = ADR$ELEMENT + 1;
  144   2           DO B1 = 1 TO NUM$MSG;   /* CHECK AMOUNT MSGS */
  145   3              IF MSG$ID = ELEMENT THEN DO;
  147   4                 ADR$ELEMENT = ADR$ELEMENT + 1 ;
  148   4                 ELEMENT = ELEMENT + 1 ; /* INCRE # RECEIVED */
  149   4                 ADR$ELEMENT = ADR$ELEMENT + 1 ;
  150   4                 CALL VECTOR$SUB(.TEMP$CLOCK1,.DTA(1),.TEMP$CLOCK2);
  151   4                 CALL VECTOR$ADD(.TEMP$CLOCK2,ADR$ELEMENT,ADR$ELEMENT);
  152   4                 RETURN TRUE;
  153   4                 END;
  154   3              ELSE ADR$ELEMENT = ADR$ELEMENT + 6; /* POINT TO NEXT */
  155   3              END;
  156   2           RETURN FALSE;   /* INVALID INPUT MSGS */
  157   2           END CORRECT$INPUT$MSG;
```

```
                $EJECT
                /*
                **************************************************************
                **************************************************************

                CORRECT$ID: CHECK IF TASK DEMANDING TASK IS CORRECT
                INPUT: ID - DEMANDING TASK NUMBER.
                       MAX - MAXIMUM OF TASKS FOR SERVICE.
                       ADR$ELEMENT - POINTER TO ELEMENT NUMBER IN ACTIVATION RECORD.
                       STEP - ACTIVATION RECORD SIZE.
                **************************************************************
                */
 158   1        CORRECT$ID: PROCEDURE (ID,MAX,ADR$ELEMENT,STEP) BYTE;
 159   2            DCL (ID,MAX) BYTE, (ADR$ELEMENT,STEP) ADDRESS;
 160   2            DCL (ELEMENT BASED ADR$ELEMENT) BYTE;

 161   2            DO B1 = 1 TO MAX;    /* LOOK AMOUNT ALL ALLOWED */
 162   3               IF ID = ELEMENT THEN DO;
 164   4                MBK = B1 - 1 ; /* SET POINTER */
 165   4                RETURN TRUE;
 166   4                END;
 167   3               ELSE ADR$ELEMENT = ADR$ELEMENT + STEP; /* POINT TO NEXT */
 168   3               END;
 169   2            RETURN FALSE;    /* INVALID TASK ID */
 170   2            END CORRECT$ID;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************

                PERIODIC$MOD: TAKES CARE OF ANY PERIODIC TASK SCHEDULED
                ***************************************************************
                */
171    1        PERIODIC$MOD: PROCEDURE;

172    2            IF NOT CORRECT$ID(B3,NUM$PER$TASK,.DBP(0).NUMBER,
                        PER$BLOCK$LEN) THEN DO;
174    3                CALL ERROR(.PERIODIC$MOD,1,1);
175    3                RETURN;
176    3                END;
177    2            CALL COPY$CLOCK(.TEMP$CLOCK1);  /* SAVE ENTRY TIME */
178    2            CALL EMULATE(.DBP(MBK).DELAY);
179    2            CALL SEND$MSGS(.DBP(MBK).NUM$MSG$OUT);
180    2            RETURN;
181    2            END PERIODIC$MOD;
```

```
                $EJECT
                /*
                ***************************************************************
                ***************************************************************

                DEMAND$MOD: TAKES CARE OF ANY DEMAND TASK SCHEDULED
                :*************************************************************
                */
182   1         DEMAND$MOD: PROCEDURE;

183   2             IF MSG(MN) = 0 THEN RETURN;

185   2             IF NOT CORRECT$ID(MSG(MN),NUM$DEM$TASK,.DBD(0).NUMBER,
                        DEM$BLOCK$LEN) THEN DO;
187   3                 CALL ERROR(.DEMAND$MOD,1,2);
188   3                 RETURN;
189   3                 END;
190   2             CALL COPY$CLOCK(.TEMP$CLOCK1);  /* GET ENTRY TIME */
191   2             IF NOT CORRECT$INPUT$MSG(DTA(0),.DBD(MBK).NUM$MSG$IN)
                      THEN DO;
193   3                 CALL ERROR(.DEMAND$MOD,1,3);
194   3                 RETURN;
195   3                 END;
196   2             IF DBD(MBK).STAT$NUM <= 1
                      THEN DO;
198   3                 DBD(MBK).STAT$NUM = DBD(MBK).NUM$OF$STAT ;
199   3                 CALL EMULATE(.DBD(MBK).DELAY);
200   3                 CALL SEND$MSGS(.DBD(MBK).NUM$MSG$OUT);
201   3                 END;
202   2                 ELSE DBD(MBK).STAT$NUM = DBD(MBK).STAT$NUM - 1 ;
203   2             RETURN;
204   2             END DEMAND$MOD;
```

```
                $EJECT
                /*
                **************************************************************
                **************************************************************

                DATA$FILLER: RECEIVES INFORMATION ABOUT NETWORK DATA
                **************************************************************
                */
205   1         DATA$FILLER: PROCEDURE;

206   2             IF MSG(MN) = 0 THEN RETURN;      /* NO USE */

208   2             IF MSG(MN) > 17 THEN DO;
210   3                          CALL ILLEGALMSG;
211   3                          RETURN;
212   3                          END;
213   2             DO CASE MSG(MN);

214   3                 RETURN;                    /* MN 0  */
215   3                 NUM$DEM$TASK = DTA(0);           /* MN 1  */
216   3                 DBP(DTA(0)).NUMBER = DTA(1);        /* MN 2  */
217   3                 DBP(DTA(0)).DELAY = DTA(1);      /* MN 3  */
218   3                 CALL MOVE(2,.DTA(1),.DBP(DTA(0)).PERIOD(2));
219   3                 DBP(DTA(0)).NUM$MSG$OUT = DTA(1);   /* MN 5  */
220   3                 CALL MOVE(4,.DTA(2),.DBP(DTA(0)).MSGS$DATA(DTA(1)));
221   3                 DBD(DTA(0)).NUMBER = DTA(1);        /* MN 7  */
222   3                 DBD(DTA(0)).DELAY = DTA(1);      /* MN 8  */
223   3                 DBD(DTA(0)).NUM$OF$STAT,DBD(DTA(0)).STAT$NUM=DTA(1);
224   3                 DBD(DTA(0)).NUM$MSG$OUT = DTA(1);   /* MN 10 */
225   3                 CALL MOVE(4,.DTA(2),.DBD(DTA(0)).MSGS$DATA(DTA(1)));
226   3                 DBD(DTA(0)).NUM$MSG$IN = DTA(1);    /* MN 12 */
227   3                 DBD(DTA(0)).MSG$IN$DATA(DTA(1)) =DTA(2);/* MN 13 */
228   3                 DO B1 = 1 TO NUM$PER$TASK ;      /* MN 14 */
229   4                    DBP(B1-1).INDEX = PERACT(.PERIODIC$MOD,.DBP(B1-1).PERIOD);
230   4                    END;
231   3                 CALL CLEAR$DATA(.NUM$PER$TASK,.MOD$BLOCK);
```

```
232   3               NUM$PER$TASK = DTA(0);          /* MN 16 */
233   3               DO B1 = 1 TO NUM$PER$TASK ;     /* MN 17 */
234   4                   CALL PERSUSP(DBP(B1-1).INDEX);
235   4                   END;
236   3               END;
237   2           RETURN;
238   2           END DATA$FILLER;
            /*
            ***************************************************************
            ***************************************************************

            SET$MODS: SETS DEMEND EMULATOR AND DATA FILEER AS SYSTEM MODULES
            ***************************************************************
            */
239   1       SET$MODS: PROCEDURE PUBLIC;

240   2           CALL ENTER$MSG$MOD(1,.DEMAND$MOD);
241   2           CALL ENTER$MSG$MOD(9,.DEMAND$MOD);
242   2           CALL ENTER$MSG$MOD(17,.DEMAND$MOD);
243   2           CALL ENTER$MSG$MOD(2,.DATA$FILLER);
244   2           CALL ENTER$MSG$MOD(10,.DATA$FILLER);
245   2           CALL ENTER$MSG$MOD(18,.DATA$FILLER);
246   2           CALL UPDSTAT(1,3);CALL UPDSTAT(9,3);CALL UPDSTAT(17,3);
249   2           CALL UPDSTAT(2,3);CALL UPDSTAT(10,3);CALL UPDSTAT(18,3);
252   2           CALL FILL(.ID$TBL,.SINK,' ');
253   2           RETURN;
254   2           END SET$MODS;



255   1       END EMULA1;
```

239

MODULE INFORMATION:

```
     CODE AREA SIZE      = 0650H   1616D
     VARIABLE AREA SIZE  = 000FH     15D
     MAXIMUM STACK SIZE  = 0006H      6D
     511 LINES READ
     0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE EMULA2
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:EMULA2.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            EMULA2: DO;
                /*
                ****************************************************************
                ****************************************************************

                THIS PROGRAM CONTAINS PROCEDURES FOR
                  CONTROLLING THE EMULATION SEQUENCE.
                0****************************************************************
                */
                $ INCLUDE (:F1:EMUDTP.SRC)
          =
          =     $ INCLUDE (:F1:SYDATE.SRC)
          =1    /*
          =1                  UPDATE: 4
          =1    */
          =1    /*
          =1    ****************************************************************
          =1    ****************************************************************
          =1    **
          =1                   SYSTEM DATA DECLARATIONS
          =1    ***
          =1    */
   2    1 =1    DECLARE LIT LITERALLY 'LITERALLY',
          =1            DCL LIT 'DECLARE';
          =1
   3    1 =1    DCL TRUE LIT '0FFH',
          =1        FALSE LIT '0',
          =1        RMN LIT '0',
          =1        SMN LIT '1',
          =1        MN LIT '2',
```

```
      =1      ML LIT '3',
      =1      MAXCPTR LIT '3',          /* MAX NUMBER OF COMPUTERS */
      =1      MAXMOD LIT '8',           /* MAX NUMBER OF MODULES IN 1 CPTR */
      =1      MAXSYSMOD LIT '24',       /* MAX NUMBER OF MODULES IN SYSTEM */
      =1      MSGBUFLEN LIT '512',      /* LENGTH OF MSG BUFFER */
      =1      MAXPRIOR LIT '8',         /* MAX NUMBER OF PRIORITY CALLS */
      =1      MAXPER LIT '8',           /* MAX NUMBER OF PERIODIC CALLS */
      =1      BEGINCM LIT '0F400H',     /* BEGIN OF COMMON MEMORY */
      =1      INTVECTOR LIT '03000H',   /* INTERRUPT VECTOR */
      =1      CPTR LIT '1',             /* NUMBER OF HOST COMPUTER */
      =1      FIRSTMN LIT '0',          /* MN OF FIRST MODULE IN CPTR */
      =1      LASTMN LIT 'FIRSTMN+7',   /* MN OF LAST MODULE IN CPTR */
      =1      EX   LIT 'FIRSTMN',       /* MN OF EXEC MODULE */
      =1      MSG$TBL$ADR LIT '0F660H';/* MSG. ENTRY TABLE ADDRESS */
      =1
      =1  /*
      =1  ***
      =1  */
      =1
4   1 =1  DCL (B1,B2,B3) BYTE EXTERNAL,
      =1      (A1,A2,A3,A4) ADDRESS EXTERNAL,
      =1      (BL,BU) BYTE  AT (.A4);
      =1            /*
      =1            SYSTEM WORK VARIABLES
      =1            BU(UPPER) AND BL(LOWER) OVERLAY ADDRESS VARIABLE A4
      =1            */
5   1 =1  DCL (ADRMSG,ADRMSGDATA) ADDRESS EXTERNAL,
      =1      (MSG BASED ADRMSG) (4) BYTE,
      =1      (MSGDATA BASED ADRMSGDATA) (100) BYTE;
      =1            /*
      =1            WORK AREAS FOR MSG CONTROL BLOCK (MSG) AND MSG
      =1            DATA BYTES (MSG DATA)
      =1            ADRMSG AND ADRMSGDATA ARE SET BY EXEC BEFORE THE
      =1            CALL OF A MODULES MSG HANDLER
      =1            */
6   1 =1  DCL INTMASK BYTE EXTERNAL;
```

```
        =1          /*
        =1          MASK OF CURRENTLY ACTIVATED INTERRUPTS
        =1          */
 7  1  =1  DCL CPTR$ID BYTE EXTERNAL;
        =1
        =   $ INCLUDE (:F1:SC.EXT)
        =1  /*
        =1  ***********************************************
        =1  ***********************************************
        =1
        =1  SYSTEM CALLS: EXTERNAL DECLARATIONS.
        =1  ***********************************************
        =1  */
        =1
 8  1  =1  FILL: PROCEDURE (P1,P2,P3) EXTERNAL;
 9  2  =1      DCL (P1,P2) ADDRESS, P3 BYTE;
10  2  =1      END;
        =1
11  1  =1  CLEARDATA: PROCEDURE (P1,P2) EXTERNAL;
12  2  =1      DCL (P1,P2) ADDRESS;
13  2  =1      END;
        =1
14  1  =1  COPY$CLOCK: PROCEDURE (P1) EXTERNAL;
15  2  =1      DCL P1 ADDRESS;
16  2  =1      END;
        =1
17  1  =1  PERACT: PROCEDURE (P1,P2) BYTE EXTERNAL;
18  2  =1      DCL (P1,P2) ADDRESS;
19  2  =1      END;
        =1
20  1  =1  PERSUSP: PROCEDURE (P1) EXTERNAL;
21  2  =1      DCL P1 BYTE;
22  2  =1      END;
        =1
23  1  =1  SEND: PROCEDURE (P1) BYTE EXTERNAL;
24  2  =1      DCL P1 ADDRESS;
```

```
 25   2  =1      END;
          =1
 26   1  =1  ILLEGALMSG: PROCEDURE EXTERNAL;
 27   2  =1      END;
          =1
 28   1  =1  PRIORITY: PROCEDURE (P1) EXTERNAL;
 29   2  =1      DCL P1 BYTE;
 30   2  =1      END;
          =1
 31   1  =1  ENTER$PRIOR: PROCEDURE (P1) BYTE EXTERNAL;
 32   2  =1      DCL P1 ADDRESS;
 33   2  =1      END;
          =1
 34   1  =1  REMPRIOR: PROCEDURE (P1) EXTERNAL;
 35   2  =1      DCL P1 BYTE ;
 36   2  =1      END;
          =1
 37   1  =1  SET$CDC: PROCEDURE (P1,P2) BYTE EXTERNAL;
 38   2  =1      DCL (P1,P2) ADDRESS;
 39   2  =1      END;
          =1
 40   1  =1  PRINT$ASYNC: PROCEDURE (P1,P2) EXTERNAL;
 41   2  =1      DCL P1 ADDRESS, P2 BYTE;
 42   2  =1      END;
          =1
 43   1  =1  PRINT$SYNC: PROCEDURE (P1,P2,P3) EXTERNAL;
 44   2  =1      DCL P1 ADDRESS, (P2,P3) BYTE;
 45   2  =1      END;
          =1
 46   1  =1  CON$INPUT$REQ: PROCEDURE (P1,P2) EXTERNAL;
 47   2  =1      DCL (P1,P2) BYTE;
 48   2  =1      END;
          =1
 49   1  =1  CON$LINK$REQ: PROCEDURE (P1) EXTERNAL;
 50   2  =1      DCL P1 BYTE;
 51   2  =1      END;
```

```
          =1
 52   1   =1  PAR$LINK$REQ: PROCEDURE (P1) EXTERNAL;
 53   2   =1      DCL P1 BYTE;
 54   2   =1      END;
          =1
 55   1   =1  CON$RELEASE: PROCEDURE EXTERNAL;
 56   2   =1      END;
          =1
 57   1   =1  PRINT$LINKED: PROCEDURE (P1,P2,P3,P4) EXTERNAL;
 58   2   =1      DCL P1 ADDRESS, (P2,P3,P4) BYTE;
 59   2   =1      END;
          =1
 60   1   =1  PAR$RELEASE: PROCEDURE EXTERNAL;
 61   2   =1      END;
          =1
 62   1   =1  PAGE: PROCEDURE (P1) EXTERNAL;
 63   2   =1      DCL P1 BYTE;
 64   2   =1      END;
          =1
 65   1   =1  WRITE: PROCEDURE (P1,P2,P3) EXTERNAL;
 66   2   =1      DCL P1 ADDRESS, (P2,P3) BYTE;
 67   2   =1      END;
          =1
 68   1   =1  WRITE$LINKED: PROCEDURE (P1,P2,P3,P4) EXTERNAL;
 69   2   =1      DCL P1 ADDRESS, (P2,P3,P4) BYTE;
 70   2   =1      END;
          =1
 71   1   =1  UPDSTAT: PROCEDURE (P1,P2) EXTERNAL;
 72   2   =1      DCL (P1,P2) BYTE;
 73   2   =1      END;
          =1
 74   1   =1  CONVASC: PROCEDURE (P1) EXTERNAL;
 75   2   =1      DCL P1 BYTE;
 76   2   =1      END;
          =1
 77   1   =1  DEBLK: PROCEDURE EXTERNAL;
```

```
 78   2  =1       END;
           =1
 79   1  =1   GET$NUM: PROCEDURE BYTE EXTERNAL;
 80   2  =1       END;
           =1
 81   1  =1   VECTOR$ADD: PROCEDURE (P1,P2,P3) EXTERNAL;
 82   2  =1       DCL (P1,P2,P3) ADDRESS;
 83   2  =1       END;
           =1
 84   1  =1   VECTOR$SUB: PROCEDURE (P1,P2,P3) EXTERNAL;
 85   2  =1       DCL (P1,P2,P3) ADDRESS;
 86   2  =1       END;
           =1
 87   1  =1   NUMOUT: PROCEDURE (P1,P2,P3,P4,P5) EXTERNAL;
 88   2  =1       DCL (P1,P4) ADDRESS, (P2,P3,P5) BYTE;
 89   2  =1       END;
           =1
 90   1  =1   GET$CHAR: PROCEDURE BYTE EXTERNAL;
 91   2  =1       END;
           =1
 92   1  =1   ENTER$MSG$MOD: PROCEDURE (P1,P2) EXTERNAL;
 93   2  =1       DCL P1 BYTE, P2 ADDRESS;
 94   2  =1       END;
           =1
 95   1  =1   ERROR: PROCEDURE (P1,P2,P3) EXTERNAL;
 96   2  =1       DCL P1 ADDRESS, (P2,P3) BYTE;
 97   2  =1       END;
           =
           =   /*
           =   ***************************************************
           =   ***************************************************
           =
           =   GENERAL DATA FOR TACTICAL SYSTEM EMULATOR
           =   ***************************************************
           =   */
```

```
98  1   =   DCL     EM          LIT '3',    /* EMULATOR MODULE NUMBER */
        =       MAX$MSG$OUT LIT '4',    /* MAX. NUMBER OF MSGS.   */
        =       MAX$MSG$IN  LIT '4',    /* IN AND OUT MODULES     */
        =
        =       NUM$PER$TASK    BYTE PUBLIC,    /* NUMBER OF PERIODIC TASK*/
        =       NUM$DEM$TASK    BYTE PUBLIC,    /* NUMBER OF DEMAND TASKS */
        =
        =       DATA$BLOCK$PER  (8) STRUCTURE   /* 8 PERIODIC MODULES MAX */
        =           (PERIOD (4) BYTE,   /* PERIOD (CLOCK FORMAT)  */
        =            NUMBER     BYTE,   /* TASK IDENTIFICATION    */
        =            DELAY      BYTE,   /* COMPUTATIONAL DELAY    */
        =            EXEC$NUM   ADDRESS,    /* NUMBER OF EXECUTIONS   */
        =            TIME$IN$EXEC (4) BYTE, /* TOTAL EXECUTION TIME   */
        =            NUM$MSG$OUT BYTE,  /* # OF OUTPUT LINKS      */
        =            MSGS$DATA (16) BYTE,   /* RECORD (4,4)           */
        =            INDEX      BYTE ) PUBLIC , /* PERIODIC INDEX */
        =
        =       DATA$BLOCK$DEM  (8) STRUCTURE   /* 8 DEMAND MODULES MAX   */
        =           (NUMBER     BYTE,   /* TASK IDENTIFICATION     */
        =            DELAY      BYTE,   /* COMPUTATIONAL DELAY    */
        =            EXEC$NUM   ADDRESS,    /* NUMBER OF EXECUTIONS   */
        =            TIME$IN$EXEC (4) BYTE, /* TOTAL EXECUTION TIME   */
        =            NUM$OF$STAT BYTE,  /* NUMBER OF STATES       */
        =            STAT$NUM   BYTE,   /* CURRENT STATE          */
        =            NUM$MSG$OUT BYTE,  /* # OF OUTPUT LINKS      */
        =            MSGS$DATA (16) BYTE,   /* RECORS (4,4)           */
        =            NUM$MSG$IN BYTE,   /* # OF INPUT LINKS       */
        =            MSG$IN$DATA (24) BYTE ) PUBLIC, /* RECORD (4,6)  */
        =
        =       TEMP$CLOCK1 (4) BYTE PUBLIC,
        =       TEMP$CLOCK2 (4) BYTE PUBLIC,
        =       TEMP$CLOCK3 (4) BYTE PUBLIC,
        =
        =       MSG$HEADER  (4) BYTE PUBLIC,
        =       MSG$TEXT    (5) BYTE PUBLIC,
```

```
=
=       MOD$BLOCK   BYTE PUBLIC,    /* DATA BLOCK POINTER     */
=
=       PER$BLOCK$LEN   LIT '30',
=       DEM$BLOCK$LEN   LIT '52',
=
=       ID$TBL (48) STRUCTURE   /* TABLE OF MODULES EXTERNAL IDS  */
=           ( NUMBER    BYTE,   /* TASK NUMBER  */
=             COMPT     BYTE,   /* HOST COMPUTER*/
=             INDEX     BYTE,   /* RELATIVE POSI*/
=             MSG$CNT   BYTE,   /* CURRENT MSG #*/
=             CPT$CNT   BYTE,   /* CURRENT CPTR */
=             KIND      BYTE) PUBLIC, /* TYPE   */
=
=       MSG$TBL(48) STRUCTURE   /* TABLE OF MSGS LINK RELATIONS   */
=           ( NUMBER    BYTE,   /* MESAGE NUMBER*/
=             INDEX     BYTE,   /* RELATIVE POSI*/
=             SOURSE    BYTE,   /* SOURSE TASK  */
=             SINK      BYTE,   /* SINK TASK    */
=             LENT      BYTE )  /* LENGTH   */ PUBLIC,
=
=       SINK BYTE PUBLIC;
=
```

```
                $EJECT
                $ INCLUDE (:F1:EMUMSG.TRE)
          =
          =     /*****************************************************************
          =     *****************************************************************
          =     : EMULA2 LOCAL VARIABLES. USED FOR CONTROLLING THE EMULATION.
          =     *****************************************************************
          =     */
          =
 99   1   =     DCL STATE          BYTE EXTERNAL ,/* STATE OF INPUT PROCESS */
          =         WRITING        BYTE EXTERNAL,/* ENTRY STATE FLAG */
          =         COUNTER        BYTE EXTERNAL,/* INDEX FOR COUNTER PROCEDURE  */
          =         EMULATION$TIME  ADDRESS EXTERNAL,/* TIME LIMIT FOR EMULATION  */
          =         EMU$CLOCK1 (4)  BYTE EXTERNAL,/* CLOCK SAVER NUMBER 1*/
          =         EMU$CLOCK2 (4)  BYTE EXTERNAL,/* CLOCK SAVER NUMBER 2 */
          =         EMU$CLOCK3 (4)  BYTE EXTERNAL,/* CLOCK SAVER NUMBER 3*/
          =         DATA$SET$MSG (4)    BYTE EXTERNAL,/* HEADER OF NETWORK MSG DATA   */
          =         OUT$TIME (14)   BYTE EXTERNAL,/* OUTPUT TIME MESSAGE*/
          =         READY          BYTE EXTERNAL,/* READY FOR EMULATION (FLAG)*/
          =         NUM$CPTR$USED   BYTE EXTERNAL,/* NUMBER OF COMPUTERS REQUESTED*/
          =         CPTR$IN$USE     BYTE EXTERNAL,/* CURRNT COMPUTER IN USE*/
          =         TASK$NUMBER     BYTE EXTERNAL,/* NUMBER OF TASK INFO RECEIVED */
          =         MSG$NUM      BYTE EXTERNAL,/* # OF MSG DATA RECEIVED*/
          =         PER$TASK$NUM    BYTE EXTERNAL,/* # OF PERIODIC TASK DATA REC. */
          =         DEM$TASK$NUM    BYTE EXTERNAL,/* # OF DEMAND TASK DATA REC.*/
          =         NPT            BYTE EXTERNAL,/* # PERIODIC TASK TO SIMULATE  */
          =         NMO            BYTE EXTERNAL,/* NUMBER OF MSG TO BE SENDED */
          =         MO             BYTE EXTERNAL,/* NUMBER OF MSG PROCESSED*/
          =         NDT            BYTE EXTERNAL,/* # OF DEMAND TASK TO SIMULATE*/
          =         NMI            BYTE EXTERNAL,/* NUMBER OF MSG TO BE RECEIVED */
          =         MI             BYTE EXTERNAL;/* NUMBER OF MSG RECIVED ALREADY*/
          =
100   1   =     DECLARE /* THE FOLLOWING MESSAGES TEXTS */
          =     MSG00(*) BYTE DATA (1,'EMULATOR...'),
          =     MSG01(*) BYTE DATA (0,'   NUMBER OF COMPUTERS NEEDED ?     .'),
          =     MSG02(*) BYTE DATA (1,'CONSOLE REFUSED !!'),
```

```
=    MSG03(*) BYTE DATA (1,'CONSOL RELEASED !!'),
=    MSG16(*) BYTE DATA (0,'PERIODIC DATA.',0AH,0DH,'      PERIOD (IN MS) ?.'),
=    MSG18(*) BYTE DATA (0,'DEMAND DATA.',0AH,0DH,'      DELAY (IN MS) ?.'),
=    MSG10(*) BYTE DATA(1,' ??????'),
=    MSG22(*) BYTE DATA(1,' '),
=    MSG33(*) BYTE DATA(1,'READY TO EMULATE ( G? ) !!'),
=    MSG39(*) BYTE DATA(1,'INVALID COMMAND!!!'),
=    MSG40(*) BYTE DATA(1,'END EMULATION .. STATISTICS AVAILABLE'),
=    MSG41(*) BYTE DATA(4,3,1,5),
=    MSG42(*) BYTE DATA(4,3,2,5),
=    MSG43(*) BYTE DATA(4,3,3,5),
=    MSG44(*) BYTE DATA(4,3,4,5),
=    MSG45(*) BYTE DATA(4,3,5,4),
=    MSG46(*) BYTE DATA(4,3,6,4),
=    MSG47(*) BYTE DATA(4,3,7,4),
=    MSG48(*) BYTE DATA(12,3,1,5),
=    MSG49(*) BYTE DATA(12,3,2,5),
=    MSG50(*) BYTE DATA(12,3,3,5),
=    MSG51(*) BYTE DATA(12,3,4,5),
=    MSG52(*) BYTE DATA(12,3,5,4),
=    MSG53(*) BYTE DATA(12,3,6,4),
=    MSG54(*) BYTE DATA(12,3,7,4),
=    MSG55(*) BYTE DATA(20,3,1,5),
=    MSG56(*) BYTE DATA(20,3,2,5),
=    MSG57(*) BYTE DATA(20,3,3,5),
=    MSG58(*) BYTE DATA(20,3,4,5),
=    MSG59(*) BYTE DATA(20,3,5,4),
=    MSG60(*) BYTE DATA(20,3,6,4),
=    MSG61(*) BYTE DATA(20,3,7,4),
=    MSG62(*) BYTE DATA(2,3,14,4),
=    MSG63(*) BYTE DATA(10,3,14,4),
=    MSG64(*) BYTE DATA(18,3,14,4),
=    MSG65(*) BYTE DATA(0,'EMULATION LASTED : '),
=    MSG66(*) BYTE DATA(1,'EMULATION INITIATED '),
=    MSG67(*) BYTE DATA(1,'UNABLE TO RELATE OUTPUT MESSAGES'),
=    MSG68(*) BYTE DATA(1,'NOT READY FOR EMULATION.'),
```

```
                =   MSG69(*) BYTE DATA(1,'INVALID TASK IDENTIFICATION'),
                =   MSG70(*) BYTE DATA(1,'INVALID COMPUTER NUMBER'),
                =   MSG71(*) BYTE DATA(1,'NO LINK MESSAGES !!!'),
                =   MSG80(*) BYTE DATA(0,'ENTER NEW DATA ? ( Y/N )  .');
                =
                    $ INCLUDE (:F1:EMULA2.EXT)
                =
 101    1       =   RECEIVE$INPUT: PROCEDURE EXTERNAL;
 102    2       =       END;
                =
 103    1       =   WRITE$ACK$RECEIVED: PROCEDURE EXTERNAL;
 104    2       =       END;
                =
 105    1       =   SEND$INFO: PROCEDURE (P1,P2,P3,P4,P5) BYTE EXTERNAL;
 106    2       =       DCL P3 ADDRESS, (P1,P2,P4,P5) BYTE;
 107    2       =       END;
```

```
                $EJECT
                $ INCLUDE (:F1:MODPRO.SRC)
          =     /*****************************************************
          =     *****************************************************
          =     : ENTER$MOD: USED FOR UPDATING THE MESSAGE TASK ENTRY POINT
          =       VECTOR FROM ISIS-II OPERATING SYSTEM.
          =     *****************************************************/
          =
  108   1 =     ENTER$MOD: PROCEDURE (P1,P2);
  109   2 =         DCL P1 BYTE, P2 ADDRESS;
  110   2 =         DCL MOD$ADR (MAXSYSMOD) ADDRESS AT (MSG$TBL$ADR),
          =             MOD$STA (MAXSYSMOD) BYTE    AT (BEGINCM);
          =
  111   2 =         MOD$ADR (P1) = P2;
  112   2 =         MOD$STA (P1) = 3;
  113   2 =         RETURN;
  114   2 =         END ENTER$MOD;
          =
          =     /*****************************************************
          =     *****************************************************
          =     : CLEAR$MOD: USED FOR CLEARING UP THE MESSAGE ENTRY
          =       POINT VECTOR FROM ISIS-II OPERATING SYSTEM.
          =     *****************************************************/
  115   1 =     CLEAR$MOD: PROCEDURE;
  116   2 =         DCL I BYTE AT (MSG$TBL$ADR-4),
          =               MOD$STA (MAXSYSMOD) BYTE AT(BEGINCM);
          =
  117   2 =         DO I = 0 TO LAST(MOD$STA);
  118   3 =             MOD$STA(I) = 0;
  119   3 =             END;
  120   2 =         RETURN;
  121   2 =         END CLEAR$MOD;
          =
          =
```

252

```
                $EJECT
 122    1       MSGENT4: PROCEDURE EXTERNAL;
 123    2           END;
 124    1       SET$MODS: PROCEDURE EXTERNAL;
 125    2           END;
 126    1       EXIT: PROCEDURE EXTERNAL;
 127    2           END;

                /*
                **************************************************************
                *
                GET$TASK: GET COMPUTER INDEX GIVEN TASK ID
                **************************************************************
                */
 128    1       GET$TASK: PROCEDURE (ID) BYTE;
 129    2           DCL ID BYTE;

 130    2           DO B2 = 1 TO TASK$NUMBER ;
 131    3               IF ID$TBL(B2-1).NUMBER = ID THEN RETURN B2-1;
 133    3               END;
 134    2           RETURN 0FFH;
 135    2           FND GET$TASK;
```

```
                $EJECT
                /*
                ****************************************************************

                PRINT$TIME: PRINT CONSUMED EMULATION TIME ON LINE PRINTER
                ****************************************************************
                */
136   1         PRINT$TIME: PROCEDURE;

137   2             IF MSGDATA(0) THEN DO; /* CONNECTED TO LINE PRINTER */
139   3                 CALL PAGE(EM);
140   3                 CALL WRITE(.MSG65,LENGTH(MSG65),EM);
141   3                 CALL PRINT$SYNC(.MSG65,LENGTH(MSG65),EM);
142   3                 OUT$TIME(0) = 1 ;
143   3                 CALL NUMOUT(DOUBLE(EMU$CLOCK3(0)),16,'.',.OUT$TIME(1),3);
144   3                 CALL NUMOUT(DOUBLE(EMU$CLOCK3(1)),16,'.',.OUT$TIME(4),3);
145   3                 CALL NUMOUT(DOUBLE(EMU$CLOCK3(2)),16,'.',.OUT$TIME(7),3);
146   3                 CALL NUMOUT(DOUBLE(EMU$CLOCK3(3)),16,'.',.OUT$TIME(10),3);
147   3                 OUT$TIME(13) = 'H';
148   3                 CALL WRITE(.OUT$TIME,LENGTH(OUT$TIME),EM);
149   3                 CALL PRINT$SYNC(.OUT$TIME,LENGTH(OUT$TIME),EM);
150   3                 CALL PAR$RELEASE;
151   3                 DO B1 = 1 TO NUM$CPTR$USED;
152   4                     CALL UPDSTAT(SHL(B1-1,3)+1,1);
153   4                     END;
154   3                 RETURN;
155   3                 END;
156   2             ELSE CALL PAR$LINK$REQ(EM);
157   2             RETURN;
158   2             END PRINT$TIME;
```

```
                $EJECT
                /*
                ************************************************************
                SET$OUT$MSG: SETS INFORMATION ABOUT INTERNAL MESSAGE RECEIVERS
                ************************************************************
                */
 159    1       SET$OUT$MSG: PROCEDURE PUBLIC;
 160    2           DCL RCPT BYTE;

 161    2           READY = FALSE;
 162    2           IF MSG$NUM = 0 THEN DO;
 164    3               READY = TRUE;
 165    3               CALL PRINT$LINKED(.MSG71,LENGTH(MSG71),EM,29);
 166    3               RETURN;
 167    3               END;
 168    2           DO B1 = 0 TO MSG$NUM - 1 ;
 169    3               B3 = GET$TASK(MSG$TBL(B1).SINK);
 170    3               IF B3 = 0FFH THEN DO;
 172    4                   CALL PRINT$SYNC(.MSG67,LENGTH(MSG67),EM);
 173    4                   CALL PRINT$LINKED(.MSG68,LENGTH(MSG68),EM,29);
 174    4                   RETURN;
 175    4                   END;
 176    3               RCPT = ID$TBL(B3).COMPT;
 177    3               B3 = GET$TASK(MSG$TBL(B1).SOURSE);
 178    3               DATA$SET$MSG(RMN) = ID$TBL(B3).COMPT + 2 ;
 179    3               IF ID$TBL(B3).KIND = 1 /* PERIODIC */
                            THEN B2 = SEND$INFO(6,10,.SET$OUT$MSG,6,11);
 181    3                     ELSE B2 = SEND$INFO(11,10,.SET$OUT$MSG,6,19);
 182    3               MSGDATA(0) = ID$TBL(B3).INDEX;
 183    3               MSGDATA(1) = MSG$TBL(B1).INDEX;
 184    3               MSGDATA(2) = RCPT + 1 ;
 185    3               MSGDATA(3) = MSG$TBL(B1).SINK;
 186    3               MSGDATA(4) = MSG$TBL(B1).NUMBER;
 187    3               MSGDATA(5) = MSG$TBL(B1).LENT;
 188    3               END;
 189    2           READY = TRUE;
```

```
190   2           CALL PRINT$LINKED(.MSG33,LENGTH(MSG33),EM,29);
191   2           END SET$OUT$MSG;
```

```
                $EJECT
                /*
                **********************************************************
                *
                SUBS$DATA: SUBSTITUTE INPUT DATA FOR EMULATION
                **********************************************************
                */
192   1         SUBS$DATA: PROCEDURE PUBLIC;

193   2             IF NOT GET$CHAR THEN DO;
195   3                 CALL PRINT$SYNC(.MSG00,LENGTH(MSG00),EM);
196   3                 CALL PRINT$SYNC(.MSG01,LENGTH(MSG01),EM);
197   3                 CALL CON$INPUT$REQ(EM,TRUE);
198   3                 STATE = 0;
199   3                 RETURN;
200   3                 END;
201   2             IF BL = 'T' THEN DO;
203   3                 IF NOT GET$NUM THEN DO;
205   4                     CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
206   4                     RETURN;
207   4                     END;
208   3                 B3 = GET$TASK(BL);
209   3                 IF B3 = 0FFH THEN DO;
211   4                     CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
212   4                     RETURN;
213   4                     END;
214   3                 TASK$NUMBER = B3 + 1 ;
215   3                 DATA$SET$MSG(RMN) = ID$TBL(B3).COMPT + 2 ;
216   3                 CPTR$IN$USE = ID$TBL(B3).CPT$CNT;
217   3                 MSG$NUM = ID$TBL(B3).MSG$CNT;
218   3                 IF ID$TBL(B3).KIND = 1 THEN DO;
220   4                     PER$TASK$NUM = ID$TBL(B3).INDEX + 1 ;
221   4                     CALL PRINT$SYNC(.MSG16,LENGTH(MSG16),EM);
222   4                     STATE = 7;
223   4                     END;
224   3                   ELSE  DO;
```

```
225   4                 DEM$TASK$NUM = ID$TBL(B3).INDEX + 1 ;
226   4                 CALL PRINT$SYNC(.MSG18,LENGTH(MSG18),EM);
227   4                 STATE = 16;
228   4                 END;
229   3             CALL CON$INPUT$REQ(EM,TRUE);
230   3             RETURN;
231   3             END;
232   2         IF BL = 'P' THEN DO;
234   3             CALL PRINT$LINKED(.MSG22,LENGTH(MSG22),EM,26);
235   3             RETURN;
236   3             END;
237   2         B1 = B1 - 1 ;
238   2         IF NOT GETNUM THEN DO;
240   3             CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
241   3             RETURN;
242   3             END;
243   2         CPTR$IN$USE = BL - 1 ;
244   2         CALL PRINT$LINKED(.MSG22,LENGTH(MSG22),EM,01);
245   2         RETURN;
246   2         END SUBS$DATA;
```

```
                $EJECT
                /*
                **********************************************************************

                MOVE$TASK: MOVE TASK LOCATION FROM ONE COMPUTER TO ANOTHER
                **********************************************************************
                */
247   1         MOVE$TASK : PROCEDURE PUBLIC;
248   2             DCL (TSK,REC,KND,NUM,OLD,NEW) BYTE;

249   2             IF NOT GETNUM THEN DO;
251   3                 CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
252   3                 RETURN;
253   3                 END;
254   2             TSK = BL;
255   2             B3 = GET$TASK(TSK);
256   2             IF B3 = 0FFH THEN DO;
258   3                 CALL PRINT$SYNC(.MSG69,LENGTH(MSG69),EM);
259   3                 CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
260   3                 RETURN;
261   3                 END;
262   2             IF NOT GETNUM THEN DO;
264   3                 CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
265   3                 RETURN;
266   3                 END;
267   2             REC = BL;
268   2             IF REC = ID$TBL(B3).CPT$CNT OR
                       REC < 1  OR  REC > 3  THEN DO;
270   3                 CALL PRINT$SYNC(.MSG70,LENGTH(MSG70),EM);
271   3                 CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
272   3                 RETURN;
273   3                 END;
274   2             KND = ID$TBL(B3).KIND;
275   2             REC = SHL(REC-1,3) + 4 ;
276   2             DO CASE ID$TBL(B3).CPT$CNT - 1 ;
277   3                 A1 = .MSG45;
```

```
278   3              A1 = .MSG52;
279   3              A1 = .MSG59;
280   3              END;
281   2          B2 = SEND(.A1);
282   2          MSGDATA(0) = KND;
283   2          MSGDATA(1) = TSK;
284   2          MSGDATA(2) = REC;

285   2          OLD = ID$TBL(B3).COMPT + 1 ;
286   2          NEW = REC - 3;
287   2          DO B1 = 1 TO MSG$NUM;
288   3              IF MSG$TBL(B1-1).SINK = TSK THEN DO;
290   4                  NUM = MSG$TBL(B1-1).SOURSE;
291   4                  B3 = GET$TASK(NUM);
292   4                  IF B3 <> 0FFH THEN DO;
294   5                      DO CASE ID$TBL(B3).CPT$CNT - 1;
295   6                          A1 = .MSG47;
296   6                          A1 = .MSG54;
297   6                          A1 = .MSG61;
298   6                          END;
299   5                      B2 = SEND(A1);
300   5                      MSGDATA(0) = NUM;
301   5                      MSGDATA(1) = OLD;
302   5                      MSGDATA(2) = NEW;
303   5                      END;
304   4                  END;
305   3              END;
306   2          CALL PRINT$LINKED(.MSG22,LENGTH(MSG22),EM,29);
307   2          END MOVE$TASK;
```

```
                $EJECT
                /*
                ***************************************************************

                EMU$COUNT: EMULATION COUNTER
                ***************************************************************:***
                */
308    1        EMU$COUNT: PROCEDURE;

309    2            DISABLE;
310    2            CALL PRIORITY(COUNTER);
311    2            ENABLE;
312    2            RETURN;
313    2            END EMU$COUNT;
                /*
                ***************************************************************

                EMU$COUNTER: COUNTS UP UNTIL END OF EMULATION
                ***************************************************************
                */
314    1        EMU$COUNTER: PROCEDURE;
315    2            DCL STOP (4) BYTE DATA(EM,EM,30,4);

316    2            IF EMULATION$TIME = 0 THEN DO;
318    3                B2 = SEND(.STOP);
319    3                CALL REMPRIOR(COUNTER);
320    3                RETURN;
321    3                END;
322    2            IF EMULATION$TIME < 50 THEN EMULATION$TIME = 50;
324    2            EMULATION$TIME = EMULATION$TIME- 50;
325    2            IF NOT SET$CDC(0D203H,.EMU$COUNT) /* SET FOR 50 MS */
                        THEN CALL ERROR(.EMU$COUNTER,6,30);
327    2            RETURN;
328    2            END EMU$COUNTER;
```

```
                $EJECT
                /*
                ****************************************************************

                START$EMU: INITIALIZE EMULATION
                ****************************************************************
                */
329    1        START$EMU: PROCEDURE PUBLIC;

330    2            IF NOT READY THEN DO;
332    3                CALL PRINT$LINKED(.MSG68,LENGTH(MSG68),EM,29);
333    3                RETURN;
334    3                END;
335    2            DO B1 = 1 TO NUM$CPTR$USED ;
336    3                DO CASE B1-1;
337    4                    B2 = SEND(.MSG62);
338    4                    B2 = SEND(.MSG63);
339    4                    B2 = SEND(.MSG64);
340    4                    END;
341    3                END;
342    2            CALL PRINT$SYNC(.MSG66,LENGTH(MSG66),EM);
343    2            IF EMULATION$TIME = 0 THEN DO;
345    3                CALL COPY$CLOCK(.EMU$CLOCK1);
346    3                RETURN;
347    3                END;
348    2            COUNTER = ENTER$PRIOR(.EMU$COUNTER);
349    2            IF NOT SET$CDC(0D203H,.EMU$COUNT) /* SET 50 MS */
                        THEN CALL ERROR(.START$EMU,6,31);
351    2            CALL COPY$CLOCK(.EMU$CLOCK1);
352    2            RETURN;
353    2            END START$EMU;
```

```
                $EJECT
                /*
                **************************************************************

                SEND$WRITE$MSGS: SENDS REQUEST TO OUTPUT INPUT DATA
                **************************************************************
                */
354   1         SEND$WRITE$MSGS: PROCEDURE (H,PRN,CPT);
355   2             DCL (H,PRN,CPT) BYTE;

356   2             IF NUM$CPTR$USED = 0 THEN DO;
358   3                 CALL PRINT$LINKED(.MSG10,LENGTH(MSG10),EM,29);
359   3                 RETURN;
360   3                 END;
361   2             IF CPT > NUM$CPTR$USED THEN DO;
363   3                 CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
364   3                 RETURN;
365   3                 END;
366   2             A1 = .MSG41 + DOUBLE(SHL(PRN,2));
367   2             DO CASE CPT;
368   3                 DO B2 = 1 TO NUM$CPTR$USED;
369   4                     B3 = SEND(A1);
370   4                     MSGDATA(0) = H ;
371   4                     A1 = A1 + 28;
372   4                     END;
373   3                 DO; B3 = SEND(A1);
375   4                     MSGDATA(0) = H ;
376   4                     END;
377   3                 DO; A1 = A1 + 28;
379   4                     B3 = SEND(A1);
380   4                     MSGDATA(0) = H ;
381   4                     END;
382   3                 DO; A1 = A1 + 56;
384   4                     B3 = SEND(A1);
385   4                     MSGDATA(0) = H ;
386   4                     END;
```

```
387   3             END;
388   2          WRITING = TRUE;
389   2          CALL CON$RELEASE;
390   2          RETURN;
391   2          END SEND$WRITE$MSGS;
```

```
                $EJECT
                /*
                ***************************************************************

                WRITE$DATA: WRITES INPUTED DATA ON LINE PRINTER
                ***************************************************************
                */
392   1         WRITE$DATA: PROCEDURE PUBLIC;
393   2             DCL (HARD$COPY,PRN,CPT) BYTE;

394   2             HARD$COPY = FALSE;
395   2             PRN = 2 ; CPT = 0 ;
397   2             DO WHILE 1; /* FOREVER */
398   3                 IF NOT GET$CHAR THEN DO;
400   4                     CALL SEND$WRITE$MSGS(HARD$COPY,PRN,CPT);
401   4                     RETURN;
402   4                     END;
403   3                 IF BL = 'P' THEN PRN = 0;
405   3                 IF BL = 'D' THEN PRN = 1;
407   3                 IF BL = 'S' THEN PRN = 3;
409   3                 IF BL > '0' AND BL < '4' THEN CPT = BL - '0' ;
411   3                 IF BL = 'H' THEN HARD$COPY = TRUE;
413   3                 END;
414   2             END WRITE$DATA;
                /*
                ***************************************************************

                DISCONNECT$CRT: CRT CONNECTION REQUEST HAS BEEN REFUSED
                ***************************************************************
                */
415   1         DISCONNECT$CRT: PROCEDURE;
416   2             CALL PRINT$ASYNC(.MSG03,LENGTH(MSG03));
417   2             RETURN;
418   2             END DISCONNECT$CRT;
```

```
                $EJECT
                /*
                ***************************************************************

                CHECK$SE$CONNECTION: CHECK CONNECTION WITH CONSOLE
                ***************************************************************::
                */
  419   1       CHECK$SE$CONNECTION: PROCEDURE;
  420   2           IF MSGDATA(0) THEN DO;
  422   3               CALL PRINT$SYNC(.MSG00,LENGTH(MSG00),EM);
  423   3               IF NOT WRITING THEN DO;
  425   4                   CALL PRINT$SYNC(.MSG80,LENGTH(MSG80),EM);
  426   4                   CALL CON$INPUT$REQ(EM,TRUE);
  427   4                   STATE = 32;
  428   4                   END;
  429   3               ELSE CALL PRINT$LINKED(.MSG22,LENGTH(MSG22),EM,29);
  430   3               RETURN;
  431   3               END;
  432   2           CALL PRINT$ASYNC(.MSG02,LENGTH(MSG02));
  433   2           CALL CON$LINK$REQ(EM);
  434   2           RETURN;
  435   2           END CHECK$SE$CONNECTION;
                /*
                ***************************************************************

                INIT$MU: INITIALIZE EMULATION
                ***************************************************************
                */
  436   1       INIT$EMU: PROCEDURE;
  437   2           CALL SET$MODS;
  438   2           CALL CLEAR$DATA(.STATE,.MI);
  439   2           CALL CON$LINK$REQ(EM);
  440   2           WRITING = FALSE;
  441   2           RETURN;
  442   2           END INIT$EMU;
```

```
                $EJECT
                /*
                **********************************************************************

                STOP$EMU: STOP A EMULATION
                **********************************************************************
                */
443   1         STOP$EMU: PROCEDURE;
444   2             DCL STOP$PER1 (4) BYTE DATA(2,3,17,4),
                      STOP$PER2 (4) BYTE DATA(10,3,17,4),
                      STOP$PER3 (4) BYTE DATA(18,3,17,4);

445   2             CALL COPY$CLOCK(.EMU$CLOCK2);
446   2             CALL VECTOR$SUB(.EMU$CLOCK2,.EMU$CLOCK1,.EMU$CLOCK3);
447   2             CALL PAR$LINK$REQ(EM);

448   2             DO B1 = 1 TO NUM$CPTR$USED;
449   3                 DO CASE B1-1;
450   4                     B2 = SEND(.STOP$PER1);
451   4                     B2 = SEND(.STOP$PER2);
452   4                     B2 = SEND(.STOP$PER3);
453   4                     END;
454   3                 CALL UPDSTAT(SHL(B1-1,3)+1,1);
455   3                 END;
456   2             CALL PRINT$LINKED(.MSG40,LENGTH(MSG40),3,29);
457   2             RETURN;
458   2             END STOP$EMU;
```

```
                $EJECT
                /*
                :*************************************************************
                *************************************************************

                MSGENT3: MESSAGE ENTRY POINT FOR DATA ENTRY MODULE
                *************************************************************
                */
459   1         MSGENT3: PROCEDURE;

460   2             IF MSG(MN) = 0 THEN DO;
462   3                 CALL INIT$EMU;
463   3                 RETURN;
464   3                 END;
465   2             IF MSG(MN) > 31 OR MSG(MN) < 20 THEN DO;
467   3                 CALL ILLEGALMSG;
468   3                 RETURN;
469   3                 END;
470   2             DO CASE MSG(MN) - 20;
471   3                 CALL RECEIVE$INPUT; /* MN 20 */
472   3                 CALL CHECK$SE$CONNECTION;    /* MN 21 */
473   3                 CALL DISCONNECT$CRT;         /* MN 22 */
474   3                 CALL ILLEGALMSG;
475   3                 CALL ILLEGALMSG;
476   3                 CALL ILLEGALMSG;
477   3                 CALL WRITE$ACK$RECEIVED;     /* MN 26 */
478   3                 CALL ILLEGALMSG;
479   3                 CALL PRINT$TIME;         /* MN 28 */
480   3                 CALL ILLEGALMSG;
481   3                 CALL STOP$EMU;           /* MN 30 */
482   3                 CALL CON$LINK$REQ(EM);       /* MN 31 */
483   3                 END;
484   2             RETURN;
485   2             END MSGENT3;
```

```
                $EJECT
                /*
                **********************************************************************
                **********************************************************************
                **********************************************************************
                *
                MAIN PROGRAM : EXECUTED UNDER ISIS. STABLISH MODULES INTO THE
                      SYSTEM.
                **********************************************************************
                */
 486    1           CALL CLEAR$MOD;
 487    1           CALL ENTER$MOD(3,.MSGENT3);
 488    1           CALL ENTER$MOD(4,.MSGENT4);
 489    1           CALL ENTER$MOD(12,.MSGENT4);
 490    1           CALL ENTER$MOD(20,.MSGENT4);
 491    1           CALL EXIT;

 492    1       END FMULA2;
```

269

MODULE INFORMATION:

```
     CODE AREA SIZE     = 0CCBH   3275D
     VARIABLE AREA SIZE = 04CAH   1226D
     MAXIMUM STACK SIZE = 000AH     10D
     857 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE EMULA3
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:EMULA3.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            EMULA3:
                /*****************************************************
                *****************************************************
                : THIS PROGRAM WRITES DATA AND STATISTICS
                  ON THE SYSTEM CONSOLE OR LINE PRINTER.
                *****************************************************
                */
                DO;

                $NOLIST
                $ INCLUDE (:F1:EMUMSG.TWO)
          =     /*****************************************************
          =     *****************************************************
          =     : EMULA3 LOACAL DATA. CONTAINS THE FORMATS FOR THE DISPLAYS.
          =     *****************************************************
          =     */
          =
  99    1 =     DECLARE /* THE FOLLOWING DATA */
          =         MSG00(*) BYTE DATA(0,'EMULATION ELAPSED :'),
          =         MSG01(*) BYTE DATA(1,' '),
          =         MSG02(*) BYTE DATA(0,'DATA COMPUTER .'),
          =         MSG03(*) BYTE DATA(0,'   PERIODIC TASK : '),
          =         MSG04(*) BYTE DATA(0,'   TASK #'),
          =         MSG05(*) BYTE DATA(0,'   -------'),
          =         MSG06(*) BYTE DATA(0,'    PERIOD'),
          =         MSG07(*) BYTE DATA(0,'    ------'),
          =         MSG08(*) BYTE DATA(0,'   DELAY '),
          =         MSG09(*) BYTE DATA(0,'   ----- '),
          =         MSG10(*) BYTE DATA(1,'     MSG OUT  '),
          =         MSG11(*) BYTE DATA(0,'   -----------'),
```

```
=       MSG12(*) BYTE DATA(0,'   # STATES'),
=       MSG13(*) BYTE DATA(0,'   --------'),
=       MSG14(*) BYTE DATA(0,'     MSG IN   '),
=       MSG15(*) BYTE DATA(0,'   ----------'),
=       MSG16(*) BYTE DATA(0,'   # EXECS'),
=       MSG17(*) BYTE DATA(0,'   -------'),
=       MSG18(*) BYTE DATA(0,'   T. IN EXEC'),
=       MSG19(*) BYTE DATA(0,'   ----------'),
=       MSG20(*) BYTE DATA(1,'   AVG.TIME'),
=       MSG21(*) BYTE DATA(1,'   --------'),
=       MSG22(*) BYTE DATA(0,'      NUMBER'),
=       MSG23(*) BYTE DATA(0,'      ------'),
=       MSG24(*) BYTE DATA(0,'   # RECEV.'),
=       MSG25(*) BYTE DATA(0,'   --------'),
=       MSG26(*) BYTE DATA(0,'   TOTAL DELAY'),
=       MSG27(*) BYTE DATA(0,'   -----------'),
=       MSG28(*) BYTE DATA(0,'   DEMAND TASK : '),
=       MSG29(*) BYTE DATA(1,'   MESSAGES : '),
=
=       MSG30(10) BYTE, /* TASK */
=       MSG31(11) BYTE, /* PERIOD */
=       MSG32(10) BYTE, /* DELAY */
=       MSG33(15) BYTE, /* MSGOUT */
=       MSG34(12) BYTE, /* # STAT */
=       MSG35(15) BYTE, /* MSGIN */
=       MSG36(11) BYTE, /* # EXEC */
=       MSG37(14) BYTE, /* TIME */
=       MSG38(12) BYTE, /* AVG. T */
=       MSG39(13) BYTE, /* NUMBER */
=       MSG40(12) BYTE, /* RECEIV */
=       MSG41(15) BYTE, /* T.DELAY */
=       MSG42     BYTE,
=
=       PRN$SEQUENCE BYTE,/* INDEX FOR DATA PRINT SEQUENCE */
=       HARD$COPY    BYTE,/* HARD COPY FLAG             */
=       MOD$ID       BYTE,/* MODULE IDENTIFICATION          */
```

```
=       PRN$FLAG(5)  BYTE,
=
=       MSG50(*) BYTE DATA(0,' '),
=       MSG51(*) BYTE DATA(1,'1'),
=       MSG52(*) BYTE DATA(1,'2'),
=       MSG53(*) BYTE DATA(1,'3'),
=       MSG54(*) BYTE DATA(1,'4'),
=       MSG55(*) BYTE DATA(1,'5'),
=       MSG56(*) BYTE DATA(1,'6'),
=       MSG57(*) BYTE DATA(1,'7'),
=       MSG58(*) BYTE DATA(1,'8'),
=       MSG59(*) BYTE DATA(1,'0');
```

```
                $EJECT
                /*
                ***************************************************************
                WRT: WRITES DATA ON CRT OR LINE PRINTER.
                ***************************************************************
                */
100    1        WRT: PROCEDURE (ADR,LEN,SEN);
101    2            DCL ADR ADDRESS, (LEN,SEN) BYTE;

102    2            IF HARD$COPY THEN CALL WRITE(ADR,LEN,SEN);
104    2                      ELSE CALL PRINT$SYNC(ADR,LEN,SEN);
105    2            RETURN;
106    2            END WRT;
                /*
                ***************************************************************
                WRT$LINKED: WRITES DATA ON CRT OR LINE PRINTER AND REQUEST
                      FOR TELL BACK CODE.
                ***************************************************************
                */
107    1        WRT$LINKED: PROCEDURE (ADR,LEN,SEN,TBC);
108    2            DCL ADR ADDRESS, (LEN,SEN,TBC) BYTE;

109    2            IF HARD$COPY THEN CALL WRITE$LINKED(ADR,LEN,SEN,TBC);
111    2                      ELSE CALL PRINT$LINKED(ADR,LEN,SEN,TBC);
112    2            RETURN;
113    2            END WRT$LINKED;
```

```
                $EJECT
                /*
                ************************************************************
                SET$BUFFERS: RESETS BUFFERS TO INITIAL CONDITION
                ************************************************************
                */
114     1       SET$BUFFERS: PROCEDURE;
115     2           CALL FILL(.MSG30,.MSG42,' ');
116     2           MSG30(0),MSG31(0),MSG32(0),MSG34(0),
                    MSG35(0),MSG36(0),MSG37(0),MSG39(0),
                    MSG40(0),MSG41(0) = 0 ;
117     2           MSG33(0),MSG38(0) = 1;
118     2           RETURN;
119     2           END SET$BUFFERS;
```

```
                $EJECT
                /*
                ************************************************************
                PRN$DATA$PER: PRINT PERIODIC TASK DATA
                ************************************************************
                */
120    1        PRN$DATA$PER: PROCEDURE (INDEX);
121    2            DCL INDEX BYTE;
122    2            DCL ADR$ELE ADDRESS,
                        (ELE BASED ADR$ELE) BYTE;

123    2            CALL SET$BUFFERS;

124    2            ADR$ELE = .DATA$BLOCK$PER(INDEX).NUMBER;
125    2            CALL NUMOUT(DOUBLE(ELE),10,' ',.MSG30(5),3);
126    2            CALL WRT(.MSG30,LENGTH(MSG30),MOD$ID);

127    2            ADR$ELE = ADR$ELE - 2 ; BU = ELE ;
129    2            ADR$ELE = ADR$ELE + 1 ; BL = ELE ;
131    2            CALL NUMOUT(A4,10,' ',.MSG31(5),4);
132    2            CALL WRT(.MSG31,LENGTH(MSG31),MOD$ID);

133    2            ADR$ELE = ADR$ELE + 2;
134    2            CALL NUMOUT(DOUBLE(ELE),10,' ',.MSG32(5),3);
135    2            CALL WRT(.MSG32,LENGTH(MSG32),MOD$ID);

136    2            ADR$ELE = .DATA$BLOCK$PER(INDEX).NUM$MSG$OUT;
137    2            B2 = ELE ; B3 = 4 ;
139    2            ADR$ELE = ADR$ELE + 3; /* POINT TO FIRST MSG NUMBER */
140    2            DO WHILE B2 > 0;
141    3                CALL NUMOUT(DOUBLE(ELE),10,' ',.MSG33(B3),2);
142    3                ADR$ELE = ADR$ELE + 4; /* POINT TO NEXT MSG NUMBER */
143    3                B3 = B3 + 3;
144    3                B2 = B2 - 1;
145    3                END;
146    2            CALL WRT(.MSG33,LENGTH(MSG33),MOD$ID);
```

```
                $EJECT
                /*
                ****************************************************************

                PRN$DATA$DEM: PRINT DEMAND DATA
                ****************************************************************
                */
149   1         PRN$DATA$DEM: PROCEDURE (INDEX);
150   2             DCL INDEX BYTE;
151   2             DCL ADR$ELE ADDRESS,
                        (ELE BASED ADR$ELE) BYTE;

152   2             CALL SET$BUFFERS;

153   2             ADR$ELE = .DATA$BLOCK$DEM(INDEX).NUMBER;
154   2             CALL NUMOUT(DOUBLE(ELE),10,' ',.MSG30(5),3);
155   2             CALL WRT(.MSG30,LENGTH(MSG30),MOD$ID);

156   2             ADR$ELE = ADR$ELE + 1;
157   2             CALL NUMOUT(DOUBLE(ELE),10,' ',.MSG32(5),3);
158   2             CALL WRT(.MSG32,LENGTH(MSG32),MOD$ID);

159   2             ADR$ELE = ADR$ELE + 7;
160   2             CALL NUMOUT(DOUBLE(ELE),10,' ',.MSG34(7),3);
161   2             CALL WRT(.MSG34,LENGTH(MSG34),MOD$ID);

162   2             ADR$ELE = .DATA$BLOCK$DEM(INDEX).NUM$MSG$IN;
163   2             B2 = ELE ;
164   2             B3 = 4;
165   2             ADR$ELE = ADR$ELE + 1;
166   2             DO WHILE B2 > 0;
167   3                 CALL NUMOUT(DOUBLE(ELE),10,' ',.MSG35(B3),2);
168   3                 B2 = B2 - 1;
169   3                 B3 = B3 + 3 ;
170   3                 ADR$ELE = ADR$ELE + 6;
171   3                 END;
```

276

```
172   2           CALL WRT(.MSG35,LENGTH(MSG35),MOD$ID);

173   2           ADR$ELE = .DATA$BLOCK$DEM(INDEX).NUM$MSG$OUT;
174   2           B2 = ELE;
175   2           B3 = 4;
176   2           ADR$ELE = ADR$ELE + 3; /* POINT TO FIRST MSF NUMBER */
177   2           DO WHILE B2 > 0;
178   3               CALL NUMOUT(DOUBLE(ELE),10,' ',.MSG33(B3),2);
179   3               ADR$ELE = ADR$ELE + 4; /* POINT TO NEXT MSG NUMBER */
180   3               B2 = B2 - 1;
181   3               B3 = B3 + 3;
182   3               END;
183   2           CALL WRT(.MSG33,LENGTH(MSG33),MOD$ID);
184   2           RETURN;
185   2           END PRN$DATA$DEM;
```

```
                $EJECT
                /*
                ***************************************************************:

                PRN$ORDER: PRINTS SEQUENTIAL NUMBER ON LINE PRINTER
                ***************************************************************
                */
186   1         PRN$ORDER: PROCEDURE (P1);
187   2             DCL P1 BYTE;

188   2             DO CASE P1;
189   3                 CALL WRT(.MSG59,LENGTH(MSG59),MOD$ID);
190   3                 CALL WRT(.MSG51,LENGTH(MSG51),MOD$ID);
191   3                 CALL WRT(.MSG52,LENGTH(MSG52),MOD$ID);
192   3                 CALL WRT(.MSG53,LENGTH(MSG53),MOD$ID);
193   3                 CALL WRT(.MSG54,LENGTH(MSG54),MOD$ID);
194   3                 CALL WRT(.MSG55,LENGTH(MSG55),MOD$ID);
195   3                 CALL WRT(.MSG56,LENGTH(MSG56),MOD$ID);
196   3                 CALL WRT(.MSG57,LENGTH(MSG57),MOD$ID);
197   3                 CALL WRT(.MSG58,LENGTH(MSG58),MOD$ID);
198   3                 END;
199   2             RETURN;
200   2             END PRN$ORDER;
```

```
                $EJECT
                /*
                ***************************************************************:

                PRN$HEAD$PER: PRINT HEADER OF PERIODIC TASK
                ***************************************************************
                */
201   1         PRN$HEAD$PER: PROCEDURE;

202   2             CALL WRT(.MSG03,LENGTH(MSG03),MOD$ID);
203   2             CALL PRN$ORDER(NUM$PER$TASK);
204   2             CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
205   2             CALL WRT(.MSG04,LENGTH(MSG04),MOD$ID);
206   2             CALL WRT(.MSG06,LENGTH(MSG06),MOD$ID);
207   2             CALL WRT(.MSG08,LENGTH(MSG08),MOD$ID);
208   2             CALL WRT(.MSG10,LENGTH(MSG10),MOD$ID);
209   2             CALL WRT(.MSG05,LENGTH(MSG05),MOD$ID);
210   2             CALL WRT(.MSG07,LENGTH(MSG07),MOD$ID);
211   2             CALL WRT(.MSG09,LENGTH(MSG09),MOD$ID);
212   2             CALL WRT(.MSG11,LENGTH(MSG11),MOD$ID);
213   2             CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
214   2             RETURN;
215   2             END PRN$HEAD$PER;
```

```
                $EJECT
                /*
                ***************************************************************

                PRN$HEAD$DEM: PRINT HEADER OF DEMAND TASK DATA
                ***************************************************************
                */
216   1         PRN$HEAD$DEM: PROCEDURE;

217   2             CALL WRT(.MSG28,LENGTH(MSG28),MOD$ID);
218   2             CALL PRN$ORDER (NUM$DEM$TASK);
219   2             CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
220   2             CALL WRT(.MSG04,LENGTH(MSG04),MOD$ID);
221   2             CALL WRT(.MSG08,LENGTH(MSG08),MOD$ID);
222   2             CALL WRT(.MSG12,LENGTH(MSG12),MOD$ID);
223   2             CALL WRT(.MSG14,LENGTH(MSG14),MOD$ID);
224   2             CALL WRT(.MSG10,LENGTH(MSG10),MOD$ID);
225   2             CALL WRT(.MSG05,LENGTH(MSG05),MOD$ID);
226   2             CALL WRT(.MSG09,LENGTH(MSG09),MOD$ID);
227   2             CALL WRT(.MSG13,LENGTH(MSG13),MOD$ID);
228   2             CALL WRT(.MSG15,LENGTH(MSG15),MOD$ID);
229   2             CALL WRT(.MSG11,LENGTH(MSG11),MOD$ID);
230   2             CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
231   2             RETURN;
232   2             END PRN$HEAD$DEM;
```

```
                $EJECT
                /*
                ***************************************************************

                PRN$PER: WRTS PERIODIC DATA ON LINEPRINTER
                ***************************************************************:
                */
233     1       PRN$PER: PROCEDURE;
234     2           IF PRN$SEQUENCE = 0 THEN DO;
236     3               CALL PRN$HEAD$PER;
237     3               PRN$SEQUENCE = 1;
238     3               CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,1);
239     3               RETURN;
240     3               END;
241     2           IF PRN$SEQUENCE > NUM$PER$TASK THEN DO;
243     3               CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,0);
244     3               RETURN;
245     3               END;
246     2           CALL PRN$DATA$PER(PRN$SEQUENCE-1);
247     2           CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,1);
248     2           PRN$SEQUENCE = PRN$SEQUENCE + 1;
249     2           RETURN;
250     2           END PRN$PER;
```

```
                $EJECT
                /*
                **********************************************************

                PRN$DEM: WRT DEMAND TASK DATA ON LINE PRINTER
                **********************************************************
                */
  251    1      PRN$DEM: PROCEDURE;
  252    2          IF PRN$SEQUENCE = 0 THEN DO;
  254    3              CALL PRN$HEAD$DEM;
  255    3              PRN$SEQUENCE = 1;
  256    3              CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,2);
  257    3              RETURN;
  258    3              END;
  259    2          IF PRN$SEQUENCE > NUM$DEM$TASK THEN DO;
  261    3              CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,0);
  262    3              RETURN;
  263    3              END;
  264    2          CALL PRN$DATA$DEM(PRN$SEQUENCE-1);
  265    2          CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,2);
  266    2          PRN$SEQUENCE = PRN$SEQUENCE + 1 ;
  267    2          RETURN;
  268    2          END PRN$DEM;
```

```
                $EJECT
                /*
                ***************************************************************

                STT$HEAD: PRINT HEADER OF TASK STATISTICS
                ***************************************************************
                */
 269    1       STT$HEAD: PROCEDURE;

 270    2           CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
 271    2           CALL WRT(.MSG04,LENGTH(MSG04),MOD$ID);
 272    2           CALL WRT(.MSG16,LENGTH(MSG16),MOD$ID);
 273    2           CALL WRT(.MSG18,LENGTH(MSG18),MOD$ID);
 274    2           CALL WRT(.MSG20,LENGTH(MSG20),MOD$ID);
 275    2           CALL WRT(.MSG05,LENGTH(MSG05),MOD$ID);
 276    2           CALL WRT(.MSG17,LENGTH(MSG17),MOD$ID);
 277    2           CALL WRT(.MSG19,LENGTH(MSG19),MOD$ID);
 278    2           CALL WRT(.MSG21,LENGTH(MSG21),MOD$ID);
 279    2           CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
 280    2           RETURN;
 281    2           END STT$HEAD;
                /*
                ***************************************************************

                STT$HEAD$PER: PRINT HEADER OF PERIODIC TASK STATISTICS
                ***************************************************************
                */
 282    1       STT$HEAD$PER: PROCEDURE;
 283    2           CALL WRT(.MSG03,LENGTH(MSG03),MOD$ID);
 284    2           CALL PRN$ORDER(NUM$PER$TASK);
 285    2           CALL STT$HEAD;
 286    2           RETURN;
 287    2           END STT$HEAD$PER;
```

```
                $EJECT
                /*
                **********************************************************

                STT$HEAD$DEM: PRINT HEADER OF DEMAND TASK STATISTIXS
                **********************************************************
                */
288    1        STT$HEAD$DEM: PROCEDURE;
289    2            CALL WRT(.MSG28,LENGTH(MSG28),MOD$ID);
290    2            CALL PRN$ORDER(NUM$DEM$TASK);
291    2            CALL STT$HEAD;
292    2            RETURN;
293    2            END STT$HEAD$DEM;
                /*
                **********************************************************

                STT$HEAD$MSG: PRINT GEADER FOR MESSAGE STATISTICS
                **********************************************************
                */
294    1        STT$HEAD$MSG: PROCEDURE;
295    2            CALL WRT(.MSG29,LENGTH(MSG29),MOD$ID);
296    2            CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
297    2            CALL WRT(.MSG22,LENGTH(MSG22),MOD$ID);
298    2            CALL WRT(.MSG24,LENGTH(MSG24),MOD$ID);
299    2            CALL WRT(.MSG26,LENGTH(MSG26),MOD$ID);
300    2            CALL WRT(.MSG20,LENGTH(MSG20),MOD$ID);
301    2            CALL WRT(.MSG23,LENGTH(MSG23),MOD$ID);
302    2            CALL WRT(.MSG25,LENGTH(MSG25),MOD$ID);
303    2            CALL WRT(.MSG27,LENGTH(MSG27),MOD$ID);
304    2            CALL WRT(.MSG21,LENGTH(MSG21),MOD$ID);
305    2            CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
306    2            RETURN;
307    2            END STT$HEAD$MSG;
```

```
                $EJECT
                /*
                **********************************************************
                STT$DATA: PRINT TASK DATA STATISTICS
                **********************************************************
                */
308    1        STT$DATA: PROCEDURE (ADR$ELE);
309    2            DCL ADR$ELE ADDRESS;
310    2            DCL (ELE BASED ADR$ELE) BYTE,
                        ELEA2 ADDRESS,
                        (ELEA BASED ADR$ELE) ADDRESS;

311    2            CALL SET$BUFFERS;

312    2            CALL NUMOUT(DOUBLE(ELE),10,' ',.MSG30(5),3);
313    2            CALL WRT(.MSG30,LENGTH(MSG30),MOD$ID);

314    2            ADR$ELE = ADR$ELE + 2;
315    2            CALL NUMOUT(ELEA,10,' ',.MSG36(4),5);
316    2            CALL WRT(.MSG36,LENGTH(MSG36),MOD$ID);
317    2            ELEA2 = ELEA;
318    2            ADR$ELE = ADR$ELE + 4;
319    2            BU = ELE ;
320    2            ADR$ELE = ADR$ELE + 1;
321    2            BL = ELE;
322    2            CALL NUMOUT(A4,10,' ',.MSG37(7),5);
323    2            CALL WRT(.MSG37,LENGTH(MSG37),MOD$ID);

324    2            IF ELEA2 = 0 THEN DO;
326    3                CALL WRT(.MSG59,LENGTH(MSG59),MOD$ID);
327    3                RETURN;
328    3                END;
329    2            ELEA2 = A4 / ELEA2;
330    2            CALL NUMOUT(ELEA2,10,' ',.MSG38(5),5);
331    2            CALL WRT(.MSG38,LENGTH(MSG38),MOD$ID);
```

```
                $EJECT
                /*
                ***************************************************************

                STT$DATA$MSG: PRINT MSG DATA STATICTICS
                ***************************************************
                */
334    1        STT$DATA$MSG: PROCEDURE (ADR$ELE);
335    2            DCL ADR$ELE ADDRESS;
336    2            DCL (ELE BASED ADR$ELE) BYTE,
                         ELEA ADDRESS;

337    2            CALL NUMOUT(DOUBLE(ELE),10,' ',.MSG39(9),3);
338    2            CALL WRT(.MSG39,LENGTH(MSG39),MOD$ID);

339    2            ADR$ELE = ADR$ELE + 1;
340    2            ELEA = DOUBLE(ELE);
341    2            CALL NUMOUT(ELEA,10,' ',.MSG40(8),3);
342    2            CALL WRT(.MSG40,LENGTH(MSG40),MOD$ID);

343    2            ADR$ELE = ADR$ELE + 3;
344    2            BU = ELE;
345    2            ADR$ELE = ADR$ELE + 1;
346    2            BL = ELE;
347    2            CALL NUMOUT(A4,10,' ',.MSG41(7),5);
348    2            CALL WRT(.MSG41,LENGTH(MSG41),MOD$ID);

349    2            IF ELEA = 0 THEN DO;
351    3                CALL WRT(.MSG59,LENGTH(MSG59),MOD$ID);
352    3                RETURN;
353    3                END;
354    2            ELEA = A4 / ELEA ;
355    2            CALL NUMOUT(ELEA,10,' ',.MSG38(5),5);
356    2            CALL WRT(.MSG38,LENGTH(MSG38),MOD$ID);
357    2            RETURN;
358    2            END STT$DATA$MSG;
```

```
                $EJECT
                /*
                ***************************************************************::

                STT$MSG: PRINT STATISTICS OF RECEIVED MESSAGE ON ONE TASK
                ***************************************************************
                */
359   1         STT$MSG: PROCEDURE (ADR$ELE);
360   2             DCL ADR$ELE ADDRESS;
361   2             DCL (ELE BASED ADR$ELE) BYTE,
                          ELE2 BYTE;

362   2             ELE2 = ELE;
363   2             ADR$ELE = ADR$ELE + 1;
364   2             DO WHILE ELE2 <> 0;
365   3                 CALL STT$DATA$MSG(ADR$ELE);
366   3                 ADR$ELE = ADR$ELE + 6;
367   3                 ELE2 = ELE2 - 1;
368   3                 END;
369   2             RETURN;
370   2             END STT$MSG;
```

```
                $EJECT
                /*
                ***************************************************************

                PRN$STT$PER: WRITES PERIODIC TASK STATISTICS
                ***************************************************************
                */
371    1        PRN$STT$PER: PROCEDURE;

372    2            IF PRN$SEQUENCE = 0 THEN DO;
374    3                CALL STT$HEAD$PER;
375    3                PRN$SEQUENCE = 1;
376    3                CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,3);
377    3                RETURN;
378    3                END;
379    2            IF PRN$SEQUENCE > NUM$PER$TASK THEN DO;
381    3                CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,0);
382    3                RETURN;
383    3                END;
384    2            CALL STT$DATA(.DATA$BLOCK$PER(PRN$SEQUENCE-1).NUMBER);
385    2            CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,3);
386    2            PRN$SEQUENCE = PRN$SEQUENCE + 1;
387    2            RETURN;
388    2            END PRN$STT$PER;
```

```
                $EJECT
                /*
                **********************************************************************

                PRN$STT$DEM: WRITES DEMAND TASK STATISTICS
                **********************************************************************
                */
389   1         PRN$STT$DEM: PROCEDURE;

390   2             IF PRN$SEQUENCE = 0 THEN DO;
392   3                 CALL STT$HEAD$DEM;
393   3                 PRN$SEQUENCE = 1;
394   3                 CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,4);
395   3                 RETURN;
396   3                 END;
397   2             IF PRN$SEQUENCE > NUM$DEM$TASK THEN DO;
399   3                 CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,0);
400   3                 RETURN;
401   3                 END;
402   2             CALL STT$DATA(.DATA$BLOCK$DEM(PRN$SEQUENCE-1).NUMBER);
403   2             CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,4);
404   2             PRN$SEQUENCE = PRN$SEQUENCE + 1;
405   2             RETURN;
406   2             END PRN$STT$DEM;
```

```
                $EJECT
                /*
                ***************************************************************

                PRN$STT$MSG: WRITES MESSAGES STATISTICS ON LINE PRINTER
                ***************************************************************
                */
407   1         PRN$STT$MSG: PROCEDURE;

408   2             IF PRN$SEQUENCE = 0 THEN DO;
410   3                 CALL STT$HEAD$MSG;
411   3                 PRN$SEQUENCE = 1;
412   3                 CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,5);
413   3                 RETURN;
414   3                 END;
415   2             IF PRN$SEQUENCE > NUM$DEM$TASK THEN DO;
417   3                 CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,0);
418   3                 RETURN;
419   3                 END;
420   2             CALL STT$MSG(.DATA$BLOCK$DEM(PRN$SEQUENCE-1).NUM$MSG$IN);
421   2             CALL WRT$LINKED(.MSG50,LENGTH(MSG50),MOD$ID,5);
422   2             PRN$SEQUENCE = PRN$SEQUENCE + 1;
423   2             RETURN;
424   2             END PRN$STT$MSG;
                /*
                ***************************************************************

                RESET$WRITER: RESETS OUTPUT MODULE DEVICE.
                ***************************************************************
                */
425   1         RESET$WRITER: PROCEDURE;

426   2             IF HARD$COPY THEN CALL PAR$RELEASE;
428   2                          ELSE CALL CON$RELEASE;
429   2             RETURN;
430   2             END RESET$WRITER;
```

290

```
                $EJECT
                /*
                ***************************************************************

                PRN: PINTS DATA OR STATISTIC AS INDICATED BY PRN$FLAG
                ***************************************************************
                */
431    1        PRN: PROCEDURE;
432    2            DCL P1 BYTE,
                        TBMSG(*) BYTE DATA(3,4,31,4);

433    2            PRN$SEQUENCE = 0;
434    2            DO P1 = 0 TO LAST(PRN$FLAG);
435    3                IF PRN$FLAG(P1) THEN DO;
437    4                    CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
438    4                    CALL WRT(.MSG01,LENGTH(MSG01),MOD$ID);
439    4                    CALL WRT(.MSG02,LENGTH(MSG02),MOD$ID);
440    4                    CALL PRN$ORDER(CPTR$ID+1);
441    4                    DO CASE P1;
442    5                        CALL PRN$PER;
443    5                        CALL PRN$DEM;
444    5                        CALL PRN$STT$PER;
445    5                        CALL PRN$STT$DEM;
446    5                        CALL PRN$STT$MSG;
447    5                        END;
448    4                    PRN$FLAG(P1) = FALSE;
449    4                    RETURN;
450    4                    END;
451    3                END;
452    2            CALL RESET$WRITER;
453    2            P1 = SEND(.TBMSG);
454    2            RETURN;
455    2            END PRN;
```

```
                $EJECT
                /*
                ****************************************************************

                REC$CO$TBCODE: RECEIVE CONSOLE TELL BACK CODE
                ****************************************************************
                */
456   1         REC$CO$TBCODE: PROCEDURE;
457   2             IF MSGDATA(0) > 5 THEN DO;
459   3                 CALL ERROR(.REC$CO$TBCODE,4,1);
460   3                 RETURN;
461   3                 END;
462   2             DO CASE MSGDATA(0);
463   3                 CALL PRN;
464   3                 CALL PRN$PER;
465   3                 CALL PRN$DEM;
466   3                 CALL PRN$STT$PER;
467   3                 CALL PRN$STT$DEM;
468   3                 CALL PRN$STT$MSG;
469   3                 END;
470   2             RETURN;
471   2             END;
```

```
                $EJECT
                /*
                **********************************************************************

                SEND$TASK: SEND TASK DATA TO AMOTHER COMPUTER
                **********************************************************************
                */
472   1         SEND$TASK: PROCEDURE (KIND,TASK$N,REC);
473   2             DCL (KIND,TASK$N,REC) BYTE;
474   2             DCL (STEP,MAX) BYTE,
                      (ADR$ELE,ADR2) ADDRESS,
                      (ELE BASED ADR$ELE) BYTE,
                      HEAD(4) BYTE;

475   2             IF KIND = 1 THEN DO ;
477   3                 ADR$ELE = .DATA$BLOCK$PER(0).NUMBER;
478   3                 STEP = PER$BLOCK$LEN;
479   3                 MAX = NUM$PER$TASK;
480   3                 ADR2 = .DATA$BLOCK$PER(NUM$PER$TASK-1);
481   3                 END;
482   2                ELSE DO;
483   3                 ADR$ELE = .DATA$BLOCK$DEM(0).NUMBER;
484   3                 STEP = DEM$BLOCK$LEN;
485   3                 MAX = NUM$DEM$TASK;
486   3                 ADR2 = .DATA$BLOCK$DEM(NUM$DEM$TASK-1);
487   3                 END;
488   2             DO B1 = 1 TO MAX;
489   3                 IF ELE = TASK$N THEN GO TO DOS;
491   3                 ADR$ELE = ADR$ELE + STEP;
492   3                 END;
493   2             RETURN;

494   2             DOS:    IF KIND = 1 THEN ADR$ELE = ADR$ELE - 4;
496   2                 HEAD(0) = REC; HEAD(1) = MOD$ID;
498   2                 HEAD(2) = 6;   HEAD(3) = STEP + 5;
500   2                 IF NOT SEND(.HEAD) THEN RETURN;
```

```
502   2             MSGDATA(0) = KIND;
503   2             CALL MOVE(STEP,ADR$ELE,ADRMSGDATA+1);

504   2          DO A1 = ADR$ELE TO ADR2-1;
505   3             CALL MOVE(STEP,A1+STEP,A1);
506   3             A1 = A1 + STEP;
507   3             END;
508   2          CALL CLEARDATA(ADR2,ADR2+STEP-1);
509   2          IF KIND = 1 THEN NUM$PER$TASK = NUM$PER$TASK - 1;
511   2                      ELSE NUM$DEM$TASK = NUM$DEM$TASK - 1;
512   2          RETURN;
513   2          END SEND$TASK;
```

```
                $EJECT
                /*
                **********************************************************

                REC$TASK: RECEIVES TASK SENDED BY ANOTHER COMPUTER
                **********************************************************:
                */
514   1         REC$TASK: PROCEDURE;

515   2             IF MSGDATA(0) = 1 THEN DO;
517   3                 IF NUM$PER$TASK = 8 THEN DO;
519   4                     CALL ERROR(.REC$TASK,4,2);
520   4                     RETURN;
521   4                     END;
522   3                 CALL MOVE(PER$BLOCK$LEN,ADRMSGDATA+1,
                            .DATA$BLOCK$PER(NUM$PER$TASK));
523   3                 NUM$PER$TASK = NUM$PER$TASK + 1;
524   3                 RETURN;
525   3                 END;
526   2                ELSE DO;
527   3                 IF NUM$DEM$TASK = 8 THEN DO;
529   4                     CALL ERROR(.REC$TASK,4,3);
530   4                     RETURN;
531   4                     END;
532   3                 CALL MOVE(DEM$BLOCK$LEN,ADRMSGDATA+1,
                            .DATA$BLOCK$DEM(NUM$DEM$TASK));
533   3                 NUM$DEM$TASK = NUM$DEM$TASK + 1;
534   3                 RETURN;
535   3                 END;
536   2             END REC$TASK;
```

```
                $EJECT
                /*
                ***************************************************
                SET$NEW$SINK: SETS NEW COMPUTER LOCATION FOR ALL MESSAGES
                    SENDERS OF TASK$N.
                ***************************************************:
                */
 537    1       SET$NEW$SINK: PROCEDURE (TASK$N,OLD,NEW);
 538    2           DCL (TASK$N,OLD,NEW) BYTE;
 539    2           DCL ADR$ELE ADDRESS,
                        (ELE BASED ADR$ELE) BYTE,
                        ELE2 BYTE;
 540    2           ADR$ELE = .DATA$BLOCK$PER(0).NUMBER;
 541    2           DO B1 = 1 TO NUM$PER$TASK;
 542    3               IF ELE = TASK$N THEN DO;
 544    4                   ADR$ELE = ADR$ELE + 8;
 545    4                   GO TO DOS;
 546    4                   END;
 547    3               ADR$ELE = ADR$ELE + PER$BLOCK$LEN;
 548    3               END;
 549    2           ADR$ELE = .DATA$BLOCK$DEM(0).NUMBER;
 550    2           DO B1 = 1 TO NUM$DEM$TASK;
 551    3               IF ELE = TASK$N THEN DO;
 553    4                   ADR$ELE = ADR$ELE + 10;
 554    4                   GO TO DOS;
 555    4                   END;
 556    3               ADR$ELE = ADR$ELE + DEM$BLOCK$LEN;
 557    3               END;
 558    2           RETURN;
 559    2           DOS:    ELE2 = ELE;
 560    2               ADR$ELE = ADR$ELE + 1;
 561    2               DO B1 = 1 TO ELE2;
 562    3                   IF ELE = OLD THEN ELE = NEW;
 564    3                   ADR$ELE = ADR$ELE + 3;
 565    3                   END;
 566    2           RETURN;
```

```
                $EJECT
                /*
                ***************************************************************

                SET$WRITER: SET OUTPUT MODULE DEVICE.
                ***************************************************************
                */
568    1        SET$WRITER: PROCEDURE;

569    2            IF MSGDATA(0) THEN DO;
571    3                    HARD$COPY = TRUE;
572    3                    CALL PAR$LINK$REQ(MOD$ID);
573    3                    END;
574    2                      ELSE DO;
575    3                    HARD$COPY = FALSE;
576    3                    CALL CON$LINK$REQ(MOD$ID);
577    3                    END;
578    2            RETURN;
579    2            END SET$WRITER;
```

```
                $EJECT
                /*
                **********************************************************
                **********************************************************

                MSGENT4: THIS MODULE ENTRY WILLBE USED BY ALL COMPUTERS
                **********************************************************
                */
580    1        MSGENT4: PROCEDURE PUBLIC;

581    2            IF MSG(MN) = 21 THEN DO;
583    3                IF MSGDATA(0) THEN CALL PRN;
585    3                            ELSE CALL SET$WRITER;
586    3                RETURN;
587    3                END;
588    2            IF MSG(MN) = 22 THEN RETURN; /* CRT RELEASED */
590    2            IF MSG(MN) = 26 THEN DO;
592    3                CALL REC$CO$TBCODE;
593    3                RETURN;
594    3                END;
595    2            IF MSG(MN) = 28 THEN DO;
597    3                IF MSGDATA(0) THEN CALL PRN;
599    3                            ELSE CALL SET$WRITER;
600    3                RETURN;
601    3                END;
602    2            IF MSG(MN) > 7 THEN DO;
604    3                CALL ILLEGALMSG;
605    3                RETURN;
606    3                END;

607    2            DO CASE MSG(MN);
608    3                MOD$ID = CPTR$ID + 4;              /* MN 0 */
609    3                DO; PRN$FLAG(0) = TRUE;       /* MN 1 */
611    4                    CALL SET$WRITER;
612    4                    END;
613    3                DO; PRN$FLAG(1) = TRUE;       /* MN 2 */
```

```
 615   4                  CALL SET$WRITER;
 616   4                  END;
 617   3              DO; PRN$FLAG(0),PRN$FLAG(1) = TRUE; /* MN 3 */
 619   4                  CALL SET$WRITER;
 620   4                  END;
 621   3              DO; PRN$FLAG(2),PRN$FLAG(3),PRN$FLAG(4) = TRUE;/*4*/
 623   4                  CALL SET$WRITER;
 624   4                  END;
 625   3              CALL SEND$TASK(MSGDATA(0),MSGDATA(1),MSGDATA(2));
 626   3              CALL REC$TASK;                  /* MN 6 */
 627   3              CALL SET$NEW$SINK(MSGDATA(0),MSGDATA(1),MSGDATA(2));
 628   3              END;
 629   2          RETURN;
 630   2          END MSGENT4;


 631   1      END EMULA3;
```

299

MODULE INFORMATION:

```
     CODE AREA SIZE     = 10E4H   4324D
     VARIABLE AREA SIZE = 00CEH    206D
     MAXIMUM STACK SIZE = 0012H     18D
     1074 LINES READ
     0 PROGRAM ERROR(S)
```

END OF PL/M-80 COMPILATION

```
ISIS-II PL/M-80 V3.0 COMPILATION OF MODULE EMULA4
NO OBJECT MODULE REQUESTED
COMPILER INVOKED BY:  PLM80 :F1:EMULA4.SRC NOOBJECT PAGELENGTH(39) PAGEWIDTH(90)


   1            EMULA4: DO;
                /*
                ***********************************************************
                ***********************************************************

                THIS PROGRAM CONTAINS PROCEDURES FOR THE INTERACTION WITH
                    THE USER.
                0***********************************************************
                */
                $NOLIST
                $ INCLUDE (:F1:EMUMSG.ONE)
          =     /***********************************************************
          =     ***********************************************************
          =     : EMULA4 LOCAL DATA. IT CONTAINS THE MESSAGES AND CONTROL VARIABLES
          =       FOR THE INTERACTION WITH THE USER.
          =     ***********************************************************
          =     */
          =
  99    1 =     DCL STATE         BYTE PUBLIC ,/* STATE OF INPUT PROCESS */
          =         WRITING       BYTE PUBLIC,/* ENTRY STATE FLAG */
          =         COUNTER       BYTE PUBLIC,/* INDEX FOR COUNTER PROCEDURE  */
          =         EMULATION$TIME  ADDRESS PUBLIC,/* TIME LIMIT FOR EMULATION  */
          =         EMU$CLOCK1 (4)  BYTE PUBLIC,/* CLOCK SAVER NUMBER 1*/
          =         EMU$CLOCK2 (4)  BYTE PUBLIC,/* CLOCK SAVER NUMBER 2 */
          =         EMU$CLOCK3 (4)  BYTE PUBLIC,/* CLOCK SAVER NUMBER 3*/
          =         DATA$SET$MSG (4)    BYTE PUBLIC,/* HEADER OF NETWORK MSG DATA   */
          =         OUT$TIME (14)   BYTE PUBLIC,/* OUTPUT TIME MESSAGE*/
          =         READY         BYTE PUBLIC,/* READY FOR EMULATION (FLAG)*/
          =         NUM$CPTR$USED   BYTE PUBLIC,/* NUMBER OF COMPUTERS REQUESTED*/
          =         CPTR$IN$USE     BYTE PUBLIC,/* CURRNT COMPUTER IN USE*/
```

```
           =        TASK$NUMBER     BYTE PUBLIC,/* NUMBER OF TASK INFO RECEIVED */
           =        MSG$NUM      BYTE PUBLIC,/* # OF MSG DATA RECEIVED*/
           =        PER$TASK$NUM    BYTE PUBLIC,/* # OF PERIODIC TASK DATA REC. */
           =        DEM$TASK$NUM    BYTE PUBLIC,/* # OF DEMAND TASK DATA REC.*/
           =        NPT          BYTE PUBLIC,/* # PERIODIC TASK TO SIMULATE  */
           =        NMO          BYTE PUBLIC,/* NUMBER OF MSG TO BE SENDED */
           =        MO           BYTE PUBLIC,/* NUMBER OF MSG PROCESSED*/
           =        NDT          BYTE PUBLIC,/* # OF DEMAND TASK TO SIMULATE*/
           =        NMI          BYTE PUBLIC,/* NUMBER OF MSG TO BE RECEIVED */
           =        MI           BYTE PUBLIC;/* NUMBER OF MSG RECIVED ALREADY*/
           =
100    1   =    DECLARE /* THE FOLLOWING MESSAGES TEXTS */
           =    MSG01(*) BYTE DATA(0,'   NUMBER OF COMPUTERS NEEDED.?(<=3) .'),
           =    MSG02(*) BYTE DATA(1,'CONSOLE CONNECTION REFUSED !!!'),
           =    MSG03(*) BYTE DATA(1,'CONSOL RELEASED !!!'),
           =    MSG05(*) BYTE DATA(1,'DATA LOADING PROCESS ..'),
           =    MSG06(*) BYTE DATA(1,'END DATA LOADING PROCESS ..'),
           =    MSG07(*) BYTE DATA(0,' LOADING COMPUTER ..'),
           =    MSG08(*) BYTE DATA(1,'   PERIODIC TASK DATA...:'),
           =    MSG09(*) BYTE DATA(0,'   NUMBER OF PERIODIC TASK (<=8) .?   .'),
           =    MSG11(*) BYTE DATA(1,'   DEMAND TASK DATA...:'),
           =    MSG12(*) BYTE DATA(0,'      PERIODIC TASK ..'),
           =    MSG13(*) BYTE DATA(1,'      END PERIODIC TASK.'),
           =    MSG14(*) BYTE DATA(0,'      ID NUMBER (<256) .?             .'),
           =    MSG15(*) BYTE DATA(1,'                       ALREADY USED !!!!'),
           =    MSG16(*) BYTE DATA(0,'      PERIOD (IN MS) .?               .'),
           =    MSG17(*) BYTE DATA(1,'      WRONG DATA !!!!!!'),
           =    MSG18(*) BYTE DATA(0,'      COMPUTATIONAL DELAY (IN MS) .?  .'),
           =    MSG19(*) BYTE DATA(0,'      NUMBER OF OUTPUT MSGS (<=4) .?  .'),
           =    MSG20(*) BYTE DATA(1,'        END OUTPUT MSGS'),
           =    MSG21(*) BYTE DATA(0,'      OUTPUT MESSAGE (##,##,##) ?  '),
           =    MSG22(*) BYTE DATA(1,' '),
           =    MSG23(*) BYTE DATA(0,'   NUMBER OF DEMAND TASK (<=8) .?     .'),
           =    MSG24(*) BYTE DATA(1,'END DATA FOR THIS COMPUTER!!'),
           =    MSG25(*) BYTE DATA(1,'      END DEMAND TASK DATA'),
           =    MSG26(*) BYTE DATA(0,'      DEMAND TASK...'),
```

```
=    MSG27(*) BYTE DATA(0,'      NUMBER OF INPUT MSGS (<=4) .?   .'),
=    MSG28(*) BYTE DATA(1,'      END INPUT MSGS.'),
=    MSG29(*) BYTE DATA(0,'       INPUT MSG ..(##) .?          '),
=    MSG30(*) BYTE DATA(0,'       NUMBER OF STATES .?            .'),
=    MSG31(*) BYTE DATA(0,'   TIME LIMIT EMULATION ( Y/N ) .?    .'),
=    MSG32(*) BYTE DATA(0,'   TIME LIMIT ( IN MS, < 65000 ) .?   .'),
=    MSG34(*) BYTE DATA(1,0DH,0AH,0AH,0AH,'POSSIBLES COMMANDS :..'),
=    MSG35(*) BYTE DATA(1,'   W <H><P/D/S><1/2/3>- WRITE DATA.'),
=    MSG36(*) BYTE DATA(1,'   E                   - EMULATE.'),
=    MSG37(*) BYTE DATA(1,'   S <T#/1/2/3/P>      - SUBSTITUTE DATA.'),
=    MSG38(*) BYTE DATA(1,'   M #<,1/,2/,3>       - MOVE TASK #.'),
=    MSG39(*) BYTE DATA(1,'INVALID COMMAND!!!'),
=    MSG50(*) BYTE DATA(0,' '),
=    MSG71(*) BYTE DATA(0,'1 :'),
=    MSG72(*) BYTE DATA(0,'2 :'),
=    MSG73(*) BYTE DATA(0,'3 :'),
=    MSG74(*) BYTE DATA(0,'4 :'),
=    MSG75(*) BYTE DATA(0,'5 :'),
=    MSG76(*) BYTE DATA(0,'6 :'),
=    MSG77(*) BYTE DATA(0,'7 :'),
=    MSG78(*) BYTE DATA(0,'8 :'),
=    MSG79(*) BYTE DATA(0,'0 :'),
=    MSG80(*) BYTE DATA(0,'ENTER NEW DATA ? ( Y/N )  .');
=
```

```
                    $EJECT
                    $ INCLUDE (:F1:EMULA4.EXT)
              =     /*****************************
              =     ******************************
              =     : EXTERNAL DECLARATIONS OF EMULA2
              =       PUBLIC PROCEDURES USED.
              =     ******************************
              =     */
              =
  101    1    =     SET$OUT$MSG: PROCEDURE EXTERNAL;
  102    2    =         END;
              =
  103    1    =     SUBS$DATA: PROCEDURE EXTERNAL;
  104    2    =         END;
              =
  105    1    =     MOVE$TASK: PROCEDURE EXTERNAL;
  106    2    =         END;
              =
  107    1    =     START$EMU: PROCEDURE EXTERNAL;
  108    2    =         END;
              =
  109    1    =     WRITE$DATA: PROCEDURE EXTERNAL;
  110    2    =         END;
```

```
                $EJECT
                /*
                ****************************************************************

                GET$NUMBER: CHECK NUMBER SENT BY INPUT MODULE
                ****************************************************************
                */
111    1        GET$NUMBER: PROCEDURE (ADR,LEN) BYTE ;
112    2            DCL ADR ADDRESS, LEN BYTE;

113    2            IF NOT GETNUM THEN DO;
115    3                CALL PRINT$SYNC(.MSG17,LENGTH(MSG17),EM);
116    3                CALL PRINT$SYNC(ADR,LEN,EM);
117    3                CALL CON$INPUT$REQ(EM,TRUE);
118    3                RETURN FALSE;
119    3                END;
120    2            RETURN TRUE;
121    2            END GET$NUMBER;
                /*
                ****************************************************************

                SEND$INFO: SEND INFORMATION TO EMULATOR
                ****************************************************************
                */
122    1        SEND$INFO: PROCEDURE (MGN,MGL,EADR,E1,E2) BYTE PUBLIC;
123    2            DCL EADR ADDRESS,(MGN,MGL,E1,E2) BYTE;

124    2            DATA$SET$MSG(MN) = MGN; /* SET MSG NUMBER */
125    2            DATA$SET$MSG(ML) = MGL; /* SET MSG LENGTH */
126    2            IF NOT SEND(.DATA$SET$MSG) THEN DO;
128    3                CALL ERROR(EADR,E1,E2);
129    3                RETURN FALSE;
130    3                END;
131    2            RETURN TRUE;
132    2            END SEND$INFO;
```

```
                $EJECT
                /*
                **********************************************************************

                QUEST: ASKS A QUESTION AND SETS NEXT STATE
                **********************************************************************
                */
133   1         QUEST: PROCEDURE (P1,P2,P3);
134   2             DCL (P2,P3) BYTE, P1 ADDRESS;

135   2             CALL PRINT$SYNC(P1,P2,EM);
136   2             CALL CON$INPUT$REQ(EM,TRUE);
137   2             STATE = P3;
138   2             RETURN;
139   2             END QUEST;
                /*
                **********************************************************************

                SAVE$TASK: SAVES INFORMATION ABOUT TASK AND SENDS IT TO
                      CORRESPONDING COMPUTER.
                **********************************************************************
                */
140   1         SAVE$TASK: PROCEDURE (P1,P2,P3,P4,P5) BYTE;
141   2             DCL (P1,P2,P3,P5) BYTE, P4 ADDRESS;
142   2             ID$TBL(TASK$NUMBER).NUMBER = BL;
143   2             ID$TBL(TASK$NUMBER).COMPT  = DATA$SET$MSG(RMN) - 2 ;
144   2             ID$TBL(TASK$NUMBER).INDEX  = P1 - 1 ;
145   2             ID$TBL(TASK$NUMBER).MSG$CNT= MSG$NUM;
146   2             ID$TBL(TASK$NUMBER).CPT$CNT= CPTR$IN$USE;
147   2             ID$TBL(TASK$NUMBER).KIND   = P2;
148   2             IF NOT SEND$INFO(P3,6,P4,6,P5) THEN RETURN FALSE;
150   2             MSGDATA(0) = P1 - 1;
151   2             MSGDATA(1) = BL;
152   2             TASK$NUMBER = TASK$NUMBER + 1;
153   2             RETURN TRUE;
154   2             END SAVE$TASK;
```

```
                $EJECT
                /*
                ****************************************************************

                SAVE$DELAY: SAVES INFORMATION ABOUT TASK DELAY AND SENDS IT TO
                        CORRESPONDING COMPUTER.
                ****************************************************************
                */
  155   1       SAVE$DELAY: PROCEDURE (P1,P2,P3,P4) BYTE;
  156   2           DCL (P1,P3,P4) BYTE, P2 ADDRESS;

  157   2           IF NOT GET$NUMBER(.MSG18,LENGTH(MSG18)) THEN RETURN FALSE;
  159   2           IF NOT SEND$INFO(P1,6,P2,6,P3) THEN RETURN FALSE;
  161   2           MSGDATA(0) = P4 - 1;
  162   2           MSGDATA(1) = BL;
  163   2           RETURN TRUE;
  164   2           END SAVE$DELAY;
                /*
                ****************************************************************

                CHECK$MSG$ID: CHEKC IF MSG NUMBER HAS BEEN USED BEFORE
                ****************************************************************
                */
  165   1       CHECK$MSG$ID: PROCEDURE(ID) BYTE;
  166   2           DCL ID BYTE;

  167   2           IF MSG$NUM = 0 THEN RETURN TRUE;
  169   2           DO B2 = 0 TO MSG$NUM - 1 ;
  170   3               IF MSG$TBL (B2).NUMBER = ID THEN RETURN FALSE;
  172   3               END;
  173   2           RETURN TRUE;
  174   2           END CHECK$MSG$ID;
```

```
                $EJECT
                /*
                ***************************************************************

                CHECK$TASK$ID: CHECK IF TASK ID HAS BEEN USED BEFORE
                ***************************************************************
                */
175   1         CHECK$TASK$ID: PROCEDURE(ID) BYTE;
176   2             DCL ID BYTE;

177   2             IF ID = 0 THEN RETURN FALSE;
179   2             DO B2 = 0 TO TASK$NUMBER - 1 ;
180   3                 IF ID$TBL(B2).NUMBER = ID THEN RETURN FALSE;
182   3                 END;
183   2             RETURN TRUE;
184   2             END CHECK$TASK$ID;
                /*
                ***************************************************************

                PRN$SEQUENCE: PRINT SEQUENCE NUMBER ON CONSOL
                ***************************************************************
                */
185   1         PRN$SEQUENCE: PROCEDURE (P1);
186   2             DCL P1 BYTE;
187   2             DO CASE P1;
188   3                 CALL PRINT$SYNC(.MSG79,LENGTH(MSG79),EM);
189   3                 CALL PRINT$SYNC(.MSG71,LENGTH(MSG71),EM);
190   3                 CALL PRINT$SYNC(.MSG72,LENGTH(MSG72),EM);
191   3                 CALL PRINT$SYNC(.MSG73,LENGTH(MSG73),EM);
192   3                 CALL PRINT$SYNC(.MSG74,LENGTH(MSG74),EM);
193   3                 CALL PRINT$SYNC(.MSG75,LENGTH(MSG75),EM);
194   3                 CALL PRINT$SYNC(.MSG76,LENGTH(MSG76),EM);
195   3                 CALL PRINT$SYNC(.MSG77,LENGTH(MSG77),EM);
196   3                 CALL PRINT$SYNC(.MSG78,LENGTH(MSG78),EM);
197   3                 END;
198   2             RETURN;
```

```
                $EJECT
                /*
                **************************************************************

                CHECK$LESS$THAN: CHECK UPPER LIMIT BOUND
                **************************************************************
                */
200    1        CHECK$LESS$THAN: PROCEDURE (P1,P2,P3,P4) BYTE;
201    2            DCL (P1,P3,P4) BYTE, P2 ADDRESS;

202    2            IF BL > P1 THEN DO;
204    3                CALL PRINT$SYNC(.MSG17,LENGTH(MSG17),EM);
205    3                CALL QUEST(P2,P3,P4);
206    3                RETURN FALSE;
207    3                END;
208    2            RETURN TRUE;
209    2            END CHECK$LESS$THAN;
                /*
                :**************************************************************

                SLAVE1: WORKS FOR STATES 10 AND 18
                **************************************************************
                */
210    1        SLAVE1: PROCEDURE (P1,P2);
211    2            DCL (P1,P2) BYTE;
212    2            MO = MO + 1;
213    2            IF MO > NMO THEN DO;
215    3                CALL PRINT$SYNC(.MSG20,LENGTH(MSG20),EM);
216    3                CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,P1);
217    3                RETURN;
218    3                END;
219    2            CALL PRINT$SYNC(.MSG21,LENGTH(MSG21),EM);
220    2            CALL PRN$SEQUENCE(MO);
221    2            CALL CON$INPUT$REQ(EM,TRUE);
222    2            STATE = P2;
223    2            RETURN;
```

```
                $EJECT
                /*
                **************************************************************

                SLAVE2: WORKS FOR STATES 11 AND 19.
                **************************************************************
                */
225   1         SLAVE2: PROCEDURE (P1,P2);
226   2             DCL (P1,P2) BYTE;
227   2             DCL (RTSK,RMGN,MGLN) BYTE;

228   2             IF NOT GET$NUMBER(.MSG21,LENGTH(MSG21)) THEN RETURN;
230   2             RTSK = BL;
231   2             IF NOT GET$NUMBER(.MSG21,LENGTH(MSG21)) THEN RETURN;
233   2             RMGN = BL;
234   2             IF NOT CHECK$MSG$ID(RMGN) THEN DO;
236   3                 CALL PRINT$SYNC(.MSG15,LENGTH(MSG15),EM);
237   3                 CALL QUEST(.MSG21,LENGTH(MSG21),P1);
238   3                 RETURN;
239   3                 END;
240   2             IF NOT GET$NUMBER(.MSG21,LENGTH(MSG21)) THEN RETURN;
242   2             IF NOT CHECK$LESS$THAN(100,.MSG21,LENGTH(MSG21),P1) THEN RETURN;
244   2             MGLN = BL;
245   2             MSG$TBL(MSG$NUM).NUMBER = RMGN;
246   2             MSG$TBL(MSG$NUM).INDEX  = SHL(MO-1,2);
247   2             MSG$TBL(MSG$NUM).SOURSE = ID$TBL(TASK$NUMBER-1).NUMBER;
248   2             MSG$TBL(MSG$NUM).SINK   = RTSK;
249   2             MSG$TBL(MSG$NUM).LENT   = MGLN;
250   2             MSG$NUM = MSG$NUM + 1;
251   2             CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,P2);
252   2             RETURN;
253   2             END SLAVE2;
```

```
                $EJECT
                /*
                ***********************************************************

                ***********************************************************
                ***********************************************************

                THE FOLLOWING PROCEDURES CARRY ON THE INTERFASE WITH THE USER
                IN SETTING DOWN THE EMULATION PARAMETERS.
                ***********************************************************
                */

                /******/
 254    1       STATE00: PROCEDURE;
 255    2           IF GETNUM THEN IF BL < 4 AND BL <> 0
                        THEN DO;
 258    3                   NUM$CPTR$USED = BL;
 259    3                   MSG$NUM,CPTR$IN$USE = 0;
 260    3                   ID$TBL(0).NUMBER = 30;
 261    3                   ID$TBL(0).COMPT = 2 ;
 262    3                   TASK$NUMBER = 1 ;
 263    3                   CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
 264    3                   CALL PRINT$LINKED(.MSG05,LENGTH(MSG05),EM,1);
 265    3                   RETURN;
 266    3                   END;
 267    2           CALL QUEST(.MSG01,LENGTH(MSG01),00);
 268    2           RETURN;
 269    2           END STATE00;
                /*******/
 270    1       STATE01: PROCEDURE;
 271    2           CPTR$IN$USE = CPTR$IN$USE + 1 ;
 272    2           IF CPTR$IN$USE > NUM$CPTR$USED THEN DO;
 274    3               CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
 275    3               CALL PRINT$LINKED(.MSG06,LENGTH(MSG06),EM,26);
 276    3               END;
 277    2              ELSE DO;
```

```
278   3                 CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
279   3                 CALL PRINT$SYNC(.MSG07,LENGTH(MSG07),EM);
280   3                 CALL PRN$SEQUENCE(CPTR$IN$USE);
281   3                 CALL PRINT$LINKED(.MSG22,LENGTH(MSG22),EM,2);
282   3                 END;
283   2             RETURN;
284   2             END STATE01;
                  /******/
285   1           STATE02: PROCEDURE;
286   2             DATA$SET$MSG (RMN) = SHL(CPTR$IN$USE-1,3) + 2 ;
287   2             DATA$SET$MSG (SMN) = EM;
288   2             IF NOT SEND$INFO(15,4,.STATE02,6,2) THEN RETURN;
290   2             CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
291   2             CALL PRINT$LINKED(.MSG08,LENGTH(MSG08),EM,3);
292   2             RETURN;
293   2             END STATE02;
                  /******/
294   1           STATE03: PROCEDURE;
295   2             CALL QUEST(.MSG09,LENGTH(MSG09),4);
296   2             RETURN;
297   2             END STATE03;
                  /******/
298   1           STATE04: PROCEDURE;
299   2             IF NOT GET$NUMBER(.MSG09,LENGTH(MSG09)) THEN RETURN;
301   2             IF NOT CHECK$LESS$THAN(8,.MSG09,LENGTH(MSG09),4) THEN RETURN;
303   2             IF BL = 0 THEN DO;
305   3                 CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
306   3                 CALL PRINT$LINKED(.MSG11,LENGTH(MSG11),EM,12);
307   3                 RETURN;
308   3                 END;
309   2             IF NOT SEND$INFO(16,5,.STATE04,6,4) THEN RETURN;
311   2             MSGDATA(0) = BL;
312   2             PER$TASK$NUM = 0 ;NPT = BL;
314   2             CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,05);
315   2             RETURN;
316   2             END STATE04;
```

311

```
                /******/
317    1        STATE05: PROCEDURE;
318    2            PER$TASK$NUM = PER$TASK$NUM + 1;
319    2            IF PER$TASK$NUM > NPT THEN DO;
321    3                CALL PRINT$SYNC(.MSG13,LENGTH(MSG13),EM);
322    3                CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
323    3                CALL PRINT$LINKED(.MSG11,LENGTH(MSG11),EM,12);
324    3                RETURN;
325    3                END;
326    2            CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
327    2            CALL PRINT$SYNC(.MSG12,LENGTH(MSG12),EM);
328    2            CALL PRN$SEQUENCE(PER$TASK$NUM);
329    2            CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
330    2            CALL QUEST(.MSG14,LENGTH(MSG14),06);
331    2            RETURN;
332    2            END STATE05;
                /******/
333    1        STATE06: PROCEDURE;
334    2            IF NOT GET$NUMBER(.MSG14,LENGTH(MSG14)) THEN RETURN;
336    2            IF NOT CHECK$TASK$ID(BL) THEN DO;
338    3                CALL PRINT$SYNC(.MSG15,LENGTH(MSG15),EM);
339    3                CALL QUEST(.MSG14,LENGTH(MSG14),06);
340    3                RETURN;
341    3                END;
342    2            IF NOT SAVE$TASK(PER$TASK$NUM,1,2,.STATE06,6) THEN RETURN;
344    2            CALL QUEST(.MSG16,LENGTH(MSG16),7);
345    2            RETURN;
346    2            END STATE06;
                /******/
347    1        STATE07: PROCEDURE;
348    2            IF NOT GET$NUMBER(.MSG16,LENGTH(MSG16)) THEN RETURN;
350    2            IF NOT SEND$INFO(4,7,.STATE07,6,7) THEN RETURN;
352    2            MSGDATA(0) = PER$TASK$NUM - 1;
353    2            MSGDATA(1) = BU;
354    2            MSGDATA(2) = BL;
355    2            CALL QUEST(.MSG18,LENGTH(MSG18),8);
```

```
356   2          RETURN;
357   2          END STATE07;
                /******/
358   1         STATE08: PROCEDURE;
359   2          IF NOT SAVE$DELAY(3,.STATE08,8,PER$TASK$NUM) THEN RETURN;
361   2          CALL QUEST(.MSG19,LENGTH(MSG19),9);
362   2          RETURN;
363   2          END STATE08;
                /******/
364   1         STATE09: PROCEDURE;
365   2          IF NOT GET$NUMBER(.MSG19,LENGTH(MSG19)) THEN RETURN;
367   2          IF NOT CHECK$LESS$THAN(4,.MSG19,LENGTH(MSG19),9) THEN RETURN;
369   2          IF NOT SEND$INFO(5,6,.STATE09,6,9) THEN RETURN;
371   2          MSGDATA(0) = PER$TASK$NUM - 1 ;
372   2          MSGDATA(1) = BL;
373   2          NMO = BL;
374   2          MO = 0;
375   2          CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,10);
376   2          RETURN;
377   2          END STATE09;
                /******/
378   1         STATE10: PROCEDURE;
379   2          CALL SLAVE1(5,11);
380   2          RETURN;
381   2          END STATE10;
                /******/
382   1         STATE11: PROCEDURE;
383   2          CALL SLAVE2(11,10);
384   2          RETURN;
385   2          END STATE11;
                /******/
386   1         STATE12: PROCEDURE;
387   2          CALL QUEST(.MSG23,LENGTH(MSG23),13);
388   2          RETURN;
389   2          END STATE12;
                /******/
```

```
390   1       STATE13: PROCEDURE;
391   2           IF NOT GET$NUMBER(.MSG23,LENGTH(MSG23)) THEN RETURN;
393   2           IF NOT CHECK$LESS$THAN(8,.MSG23,LENGTH(MSG23),13) THEN RETURN;
395   2           IF BL = 0 THEN DO;
397   3               CALL PRINT$LINKED(.MSG24,LENGTH(MSG24),EM,01);
398   3               RETURN;
399   3               END;
400   2           IF NOT SEND$INFO(1,5,.STATE13,6,13) THEN RETURN;
402   2           MSGDATA(0) = BL;
403   2           DEM$TASK$NUM = 0;
404   2           NDT = BL;
405   2           CALL PRINT$LINKED(.MSG22,LENGTH(MSG22),EM,14);
406   2           RETURN;
407   2           END STATE13;
                 /******/
408   1       STATE14: PROCEDURE;
409   2           DEM$TASK$NUM = DEM$TASK$NUM + 1 ;
410   2           IF DEM$TASK$NUM > NDT THEN DO;
412   3               CALL PRINT$SYNC(.MSG25,LENGTH(MSG25),EM);
413   3               CALL PRINT$LINKED(.MSG24,LENGTH(MSG24),EM,01);
414   3               RETURN;
415   3               END;
416   2           CALL PRINT$SYNC(.MSG26,LENGTH(MSG26),EM);
417   2           CALL PRN$SEQUENCE(DEM$TASK$NUM);
418   2           CALL PRINT$SYNC(.MSG22,LENGTH(MSG22),EM);
419   2           CALL QUEST(.MSG14,LENGTH(MSG14),15);
420   2           RETURN;
421   2           END STATE14;
                 /******/
422   1       STATE15: PROCEDURE;
423   2           IF NOT GET$NUMBER(.MSG14,LENGTH(MSG14)) THEN RETURN;
425   2           IF NOT CHECK$TASK$ID(BL) THEN DO;
427   3               CALL PRINT$SYNC(.MSG15,LENGTH(MSG15),EM);
428   3               CALL QUEST(.MSG14,LENGTH(MSG14),15);
429   3               RETURN;
430   3               END;
```

```
431   2           IF NOT SAVE$TASK(DEM$TASK$NUM,2,7,.STATE15,15) THEN RETURN;
433   2           CALL QUEST(.MSG18,LENGTH(MSG18),16);
434   2           RETURN;
435   2           END STATE15;
                 /******/
436   1       STATE16: PROCEDURE;
437   2           IF NOT SAVE$DELAY(8,.STATE16,16,DEM$TASK$NUM) THEN RETURN;
439   2           CALL QUEST(.MSG19,LENGTH(MSG19),17);
440   2           RETURN;
441   2           END STATE16;
                 /******/
442   1       STATE17: PROCEDURE;
443   2           IF NOT GET$NUMBER(.MSG19,LENGTH(MSG19)) THEN RETURN;
445   2           IF NOT CHECK$LESS$THAN(4,.MSG19,LENGTH(MSG19),17) THEN RETURN;
447   2           IF NOT SEND$INFO(10,6,.STATE17,6,17) THEN RETURN;
449   2           MSGDATA(0) = DEM$TASK$NUM -1 ;
450   2           MSGDATA(1) = BL;
451   2           NMO = BL;
452   2           MO = 0 ;
453   2           CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,18);
454   2           RETURN;
455   2           END STATE17;
                 /******/
456   1       STATE18: PROCEDURE;
457   2           CALL SLAVE1(20,19);
458   2           RETURN;
459   2           END STATE18;
                 /******/
460   1       STATE19: PROCEDURE;
461   2           CALL SLAVE2(19,18);
462   2           RETURN;
463   2           END STATE19;
                 /******/
464   1       STATE20: PROCEDURE;
465   2           CALL QUEST(.MSG27,LENGTH(MSG27),21);
466   2           RETURN;
```

```
467   2          END STATE20;
                /******/
468   1         STATE21: PROCEDURE;
469   2           IF NOT GET$NUMBER(.MSG27,LENGTH(MSG27)) THEN RETURN;
471   2           IF NOT CHECK$LESS$THAN(4,.MSG27,LENGTH(MSG27),21) THEN RETURN;
473   2           IF NOT SEND$INFO(12,6,.STATE21,6,21) THEN RETURN;
475   2           MSGDATA(0) = DEM$TASK$NUM -1 ;
476   2           MSGDATA(1) = BL;
477   2           NMI = BL;
478   2           MI = 0;
479   2           CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,22);
480   2           RETURN;
481   2           END STATE21;
                /******/
482   1         STATE22: PROCEDURE;
483   2           MI = MI + 1 ;
484   2           IF MI > NMI THEN DO;
486   3               CALL PRINT$LINKED(.MSG28,LENGTH(MSG28),EM,24);
487   3               RETURN;
488   3               END;
489   2           CALL PRINT$SYNC(.MSG29,LENGTH(MSG29),EM);
490   2           CALL PRN$SEQUENCE(MI);
491   2           CALL CON$INPUT$REQ(EM,TRUE);
492   2           STATE = 23;
493   2           RETURN;
494   2           END STATE22;
                /******/
495   1         STATE23: PROCEDURE;
496   2           IF NOT GET$NUMBER(.MSG29,LENGTH(MSG29)) THEN RETURN;
498   2           IF NOT SEND$INFO(13,7,.STATE23,6,23) THEN RETURN;
500   2           MSGDATA(0) = DEM$TASK$NUM - 1;
501   2           MSGDATA(1) = SHL(MI-1,2) + SHL(MI-1,1);
502   2           MSGDATA(2) = BL;
503   2           CALL PRINT$LINKED(.MSG50,LENGTH(MSG50),EM,22);
504   2           RETURN;
505   2           END STATE23;
```

```
                /******/
506    1        STATE24: PROCEDURE;
507    2            CALL QUEST(.MSG30,LENGTH(MSG30),25);
508    2            RETURN;
509    2            END STATE24;
                /******/
510    1        STATE25: PROCEDURE;
511    2            IF NOT GET$NUMBER(.MSG30,LENGTH(MSG30)) THEN RETURN;
513    2            IF NOT SEND$INFO(9,6,.STATE25,6,25) THEN RETURN;
515    2            MSGDATA(0) = DEM$TASK$NUM - 1;
516    2            MSGDATA(1) = BL;
517    2            CALL PRINT$LINKED(.MSG22,LENGTH(MSG22),EM,14);
518    2            RETURN;
519    2            END STATE25;
                /******/
520    1        STATE26: PROCEDURE;
521    2            CALL QUEST(.MSG31,LENGTH(MSG31),27);
522    2            RETURN;
523    2            END STATE26;
                /******/
524    1        STATE27: PROCEDURE;
525    2            IF MSGDATA(B1) = 'Y' THEN DO;
527    3                CALL QUEST(.MSG32,LENGTH(MSG32),28);
528    3                RETURN;
529    3                END;
530    2            CALL SET$OUT$MSG;
531    2            RETURN;
532    2            END STATE27;
                /******/
533    1        STATE28: PROCEDURE;
534    2            IF NOT GET$NUMBER(.MSG32,LENGTH(MSG32)) THEN RETURN;
536    2            EMULATION$TIME = A4;
537    2            CALL SET$OUT$MSG;
538    2            RETURN;
539    2            END STATE28;
                /******/
```

```
540   1       STATE29: PROCEDURE;
541   2           CALL PRINT$SYNC(.MSG34,LENGTH(MSG34),EM);
542   2           CALL PRINT$SYNC(.MSG35,LENGTH(MSG35),EM);
543   2           CALL PRINT$LINKED(.MSG36,LENGTH(MSG36),EM,30);
544   2           RETURN;
545   2           END STATE29;
              /******/
546   1       STATE30: PROCEDURE;
547   2           CALL PRINT$SYNC(.MSG37,LENGTH(MSG37),EM);
548   2           CALL QUEST(.MSG38,LENGTH(MSG38),31);
549   2           RETURN;
550   2           END STATE30;
              /******/
551   1       STATE31: PROCEDURE;
552   2           DCL EOT LIT '04H';
553   2           DCL COMM$LIST(*) BYTE DATA('WEMS',EOT);
554   2           DO B2 = 0 TO LAST(COMM$LIST);
555   3               IF MSGDATA(B1) = COMM$LIST(B2) THEN GO TO FOUND;
557   3               END;
558   2           CALL PRINT$LINKED(.MSG39,LENGTH(MSG39),EM,29);
559   2           RETURN;
560   2           FOUND:  B1 = B1 + 1;
561   2               DO CASE B2;
562   3                   CALL WRITE$DATA;
563   3                   CALL START$EMU;
564   3                   CALL MOVE$TASK;
565   3                   CALL SUBS$DATA;
566   3                   CALL CON$RELEASE;
567   3                   END;
568   2           RETURN;
569   2           END STATE31;
              /******/
570   1       STATE32: PROCEDURE;
571   2           IF NOT GET$CHAR THEN DO;
573   3               CALL QUEST(.MSG80,LENGTH(MSG80),32);
574   3               RETURN;
```

```
575   3             END;
576   2         IF BL <> 'Y' THEN DO;
578   3             CALL PRINT$LINKED(.MSG22,LENGTH(MSG22),EM,29);
579   3             RETURN;
580   3             END;
581   2         CALL QUEST(.MSG01,LENGTH(MSG01),00);
582   2         RETURN;
583   2         END STATE32;
```

```
                $EJECT
                /**
                ***************************************************************************

                RECEIVE$INPUT: RECEIVE INPUT FROM SYSTEM CONSOLE
                ***************************************************************************
                */
 584    1       RECEIVE$INPUT: PROCEDURE PUBLIC;

 585    2           B1 = 0; CALL DEBLK;
 587    2           IF MSGDATA(B1) = '%' THEN DO;
 589    3               STATE = 29;
 590    3               CALL STATE29;
 591    3               RETURN;
 592    3               END;
 593    2           IF STATE > 32 THEN DO;
 595    3               CALL ERROR(.RECEIVE$INPUT,3,1);
 596    3               RETURN;
 597    3               END;
 598    2           DO CASE STATE;
 599    3               CALL STATE00;
 600    3               CALL STATE01;
 601    3               CALL STATE02;
 602    3               CALL STATE03;
 603    3               CALL STATE04;
 604    3               CALL STATE05;
 605    3               CALL STATE06;
 606    3               CALL STATE07;
 607    3               CALL STATE08;
 608    3               CALL STATE09;
 609    3               CALL STATE10;
 610    3               CALL STATE11;
 611    3               CALL STATE12;
 612    3               CALL STATE13;
 613    3               CALL STATE14;
 614    3               CALL STATE15;
```

```
 615   3                CALL STATE16;
 616   3                CALL STATE17;
 617   3                CALL STATE18;
 618   3                CALL STATE19;
 619   3                CALL STATE20;
 620   3                CALL STATE21;
 621   3                CALL STATE22;
 622   3                CALL STATE23;
 623   3                CALL STATE24;
 624   3                CALL STATE25;
 625   3                CALL STATE26;
 626   3                CALL STATE27;
 627   3                CALL STATE28;
 628   3                CALL STATE29;
 629   3                CALL STATE30;
 630   3                CALL STATE31;
 631   3                CALL STATE32;
 632   3                END;
 633   2          END RECEIVE$INPUT;
```

```
                $EJECT
                /*
                ***************************************************************

                WRITE$ACK$RECEIVED: WRITE ACKNOWLEDGE RECEIVED FROM SYSTEM CONSOLE
                ***************************************************************
                */
 634    1       WRITE$ACK$RECEIVED: PROCEDURE PUBLIC;
 635    2           STATE = MSGDATA(0);
 636    2           CALL RECEIVE$INPUT;
 637    2           RETURN;
 638    2           END WRITE$ACK$RECEIVED;
 639    1       END EMULA4;


MODULE INFORMATION:

      CODE AREA SIZE     = 11A8H   4520D
      VARIABLE AREA SIZE = 0057H     87D
      MAXIMUM STACK SIZE = 00EH      14D
      985 LINES READ
      0 PROGRAM ERROR(S)

END OF PL/M-80 COMPILATION
```

LIST OF REFERENCES

[1] Arnold, C.R., "The need for distributed operating systems", Naval Underwater Systems Center, New London Lavoratory, New London, Connecticut 06320, 15 april 1975.

[2] Niemann, W., "A Real-Time Operating System for Single Board Computer Based Distributed Naval Tactical Data Systems", M.S. Thesis, Naval Postgraduate School, Monterey, June 1978.

[3] Naval Postgraduate School, "A study of alternatives for VSTOL computer systems", by U. Kodres, J. Butlinger, R. Hamming and C. Jones, April 1978.

[4] Breuer, M.A. Editor, "Design Automation of Digital Systems: Theory and Techniques", Prentice Hall, 1972.

[5] "SBC 80/20 and SBC 80/20-4 Single Board Computer Harware Reference Manual",Intel Corporation, 1977.

[6] INTELLEC Microcomputer Development System, "Hardware Refence Manual", Intel Corporation, November 1976.

[7] ISIS-II User's Guide, Intel Corporation, 1978.

[8] ISIS-II PL/M-80 Compiler Operator's Manual, Intel Corporation, 1977.

INITIAL DISTRIBUTION LIST

		No. copies
1.	Library, Code Ø142 Naval Postgraduate School Monterey, California 93940	2
2.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
3.	Assoc. Prof. Uno R. Kodres, Code 52Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	LTC Roger R. Schell, Code 52Sj Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
5.	LT(JG) Luis A. Guillen, Peruvian Navy Central de Procesamiento de Datos Ministerio de Marina Salaverry S/N, San Felipe Lima, Peru	1
6.	LT(JG) Javier De la Cuba, Peruvian Navy Central de Procesamiento de Datos Ministerio de Marina Salaverry S/N, San Felipe Lima, Peru	1
7.	Direccion de Instruccion de la Marina Ministerio de Marina Salaverry S/N, San Felipe Lima, Peru	1
8.	LT Felix Luna, Peruvian Navy 1149 Leahy Rd. Monterey, California 93940	1
9.	LCDR. Amrun Sehan 415 Casaverde #3 Monterey, California 93940	1
10.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2

Thesis
G8634 Guillen
c.1 A tactical system emulator for a distributed micro-computer architecture.

183383